For the Advanced Computer User

Micro/Systems Journal™

Build an S-100 to PC Bus Converter

see pages 24–30

Also in this Issue

Complete Table of Contents on Page 3

May/June 1985 Vol. 1/No. 2 $4.00 U.S.A.

MACROTECH—STILL THE S-100 PERFORMANCE PACESETTER

MI-286. Our 80286/Z80H Dual CPU Board is at least twice as fast as Compupro's 8085/88 and it's a direct replacement. The MI-286 has already become the standard by which other 80286 based systems are measured. Ask us for a complimentary Benchmark Report.

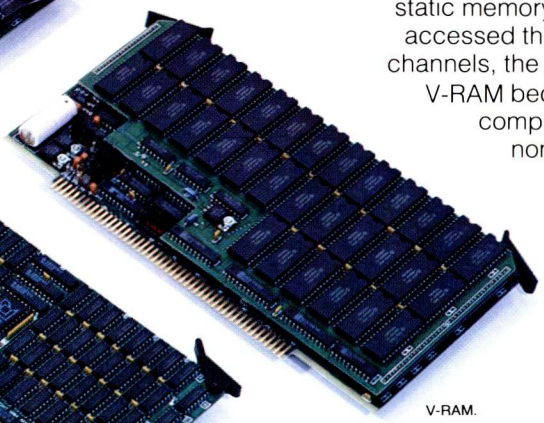
ADIT. There's nothing else like it on the market. It's an Intelligent I/O Board with its own real time firmware that lets you control up to 16 different terminals, modems or printers all from a single slot. ADIT is the performance standard in environments such as Alpha Micro where I/O speed is critical.

V-RAM. High performance Static CMOS system memory/virtual disk in either quarter or half megabyte configurations. With its on-board battery and power-fail logic, the V- RAM sets a new performance standard at conventional static memory prices. When accessed through I/O port channels, the half megabyte V-RAM becomes M Drive compatible with true non-volatile solid-state disk capability.

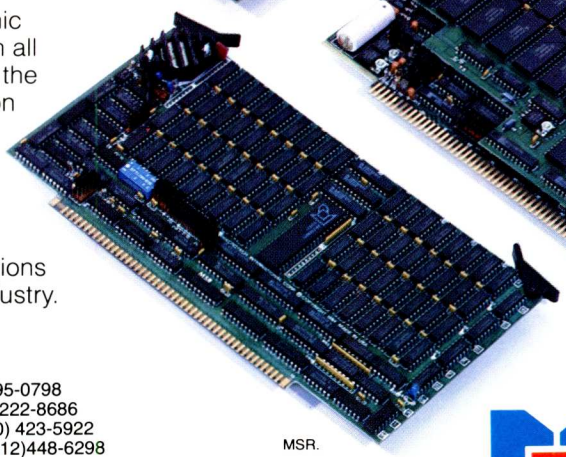
MI-286.

ADIT.

MSR. High performance and reliability in a memory so fast you won't believe it's a dynamic ram product. Compatible with all popular S-100 environments, the MSR's low power consumption and 120 nanosecond ram devices set a new standard for dynamic memory products. The MSR is available in quarter, half, one and two megabyte configurations at the lowest prices in the industry.



V-RAM.



MSR.

Dealers:
Gifford Computer Systems (415) 895-0798
Custom Computer Technology (800) 222-8686
Priority One Electronics (800) 423-5922
John D. Owens & Associates (212)448-6298
In England; Fulcrum (Europe) Ltd. (0621) 828763

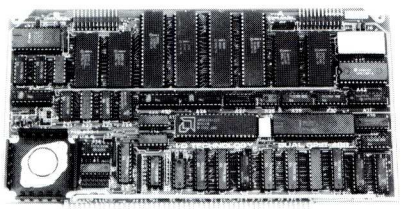
Macrotech dealers also include most Compupro Systems Centers, Heathkit Electronic Centers and Alpha Micro Dealers.

MACROTECH International Corp.
9551 Irondale Ave.
Chatsworth, CA 91311
(800) 824-3181 • in Calif. (818) 700-1501
Telex: 9109970653

INPUT/OUTPUT TECHNOLOGY, INC.

25327 Avenue Stanford, Unit 113, Valencia, CA 91355 • (805) 257-1000

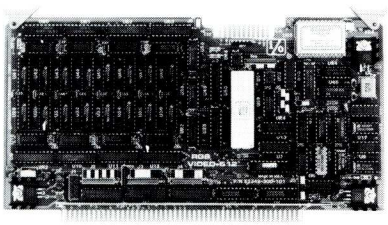
Uncompromising Additions to your S-100/IEEE-696 BUS



DUAL GPIB-488 INTERFACE BOARD

A Stand-Alone, Independently Controlled Dual Channel IEEE-488 I/O Processor. Interface Activity Modes for Controller-in-Charge, Controller Assigned or Terminal Bus Slave, and all Interface Functions are handled transparent to Host System CPU through an on-board CPU and DMA controller. User Friendly operation.

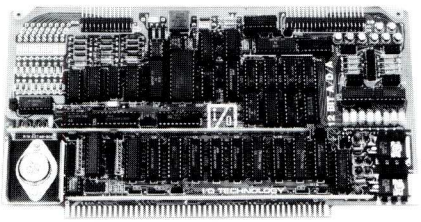
A&T, P/N 52748-800-102



RGB COLOR GRAPHICS BOARD

Programmable resolution up to 512 x 512 pixels with 4 local video planes and on-board graphics processor. Color mapper allows 16 colors from a palette of 4096. Light pen input. Plus more ...

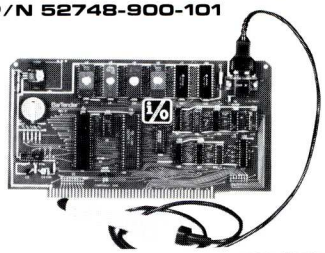
A&T, P/N 52748-300-101



12-BIT A-D-A CONVERTER BOARD

8 Channel A-D: 12 microsec. Conversion, 50KHz Sample Rate, Programmable Gains, Offset and Diff./Single Modes.
8 Channel D-A: 2 microsec. Settling, Bipolar V or Unipolar I Output. Programmable Reference levels, Dual-Ported Channel Refresh RAM. **16/8-Bit Data Transfers** via I/O or Memory Mapped

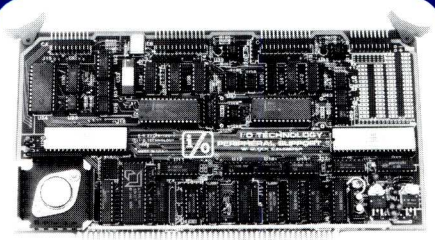
A&T, P/N 52748-900-101



BAR CODE PROCESSOR BOARD

The BarTender is a stand-alone I/O Processor that reads and prints most common Bar Codes. Includes bi-directional reading, wand interface, clock/calendar with battery. Extensive documentation and software.

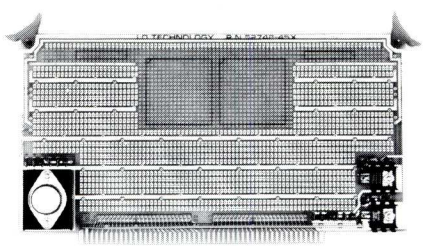
A&T,52748-500-101 Without Wand
A&T,52748-500-201 With Wand



PERIPHERAL SUPPORT BOARD

Two Serial SYNC/ASYNC Ports with RS-232, TTL or Current Loop Outputs, three 8-Bit Parallel Ports, three Timers, Real Time Clock/Calendar and Response Programmable Interrupt Controller. Small Proto Area with +5 and ±12v.

A&T, P/N 52748-150-101



MULTI-PURPOSE PROTOTYPING KIT

Industrial Quality with Plated-Thru holes for Wire-Wrap or Solder projects. Complete with +5, ±12v Regulators, Bus Bar, Filter Capacitors, and Manual.

P/N 52748-450

ALSO AVAILABLE: MULTI-FUNCTION I/O BOARD, SMART PROTOTYPING KIT, 128Kx8/64Kx16 STATIC RAM MODULE

CIRCLE 10 ON READER SERVICE CARD

SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE.

Teletek nology

One Success After Another.

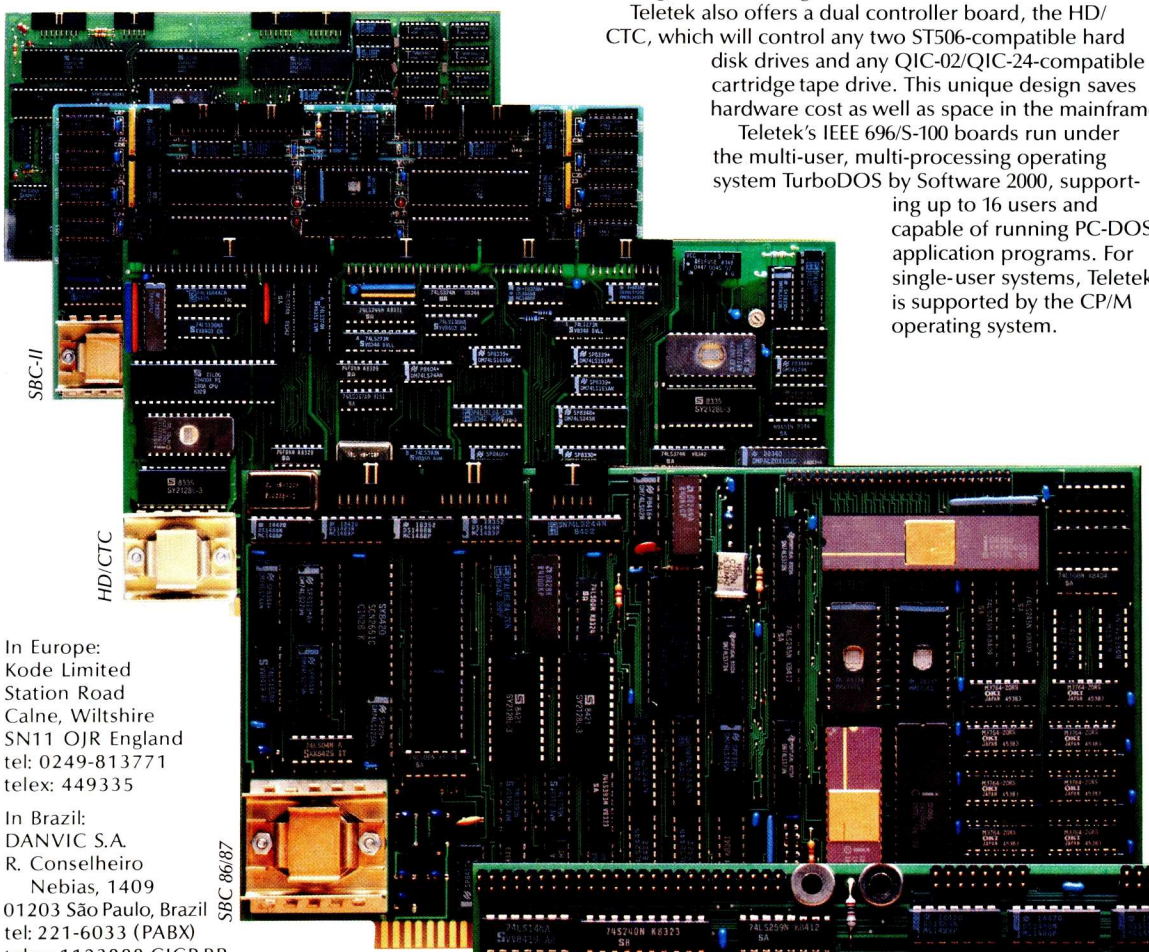
Since 1968 Teletek has been a leader in the design and manufacture of single board computers, controllers, memory boards and interface boards.

Teletek offers five distinct single board computers (SBCs), each with its own unique features, to meet the varied needs of the system integrator. Based on the 8086 16-bit and Z80 8-bit microprocessors, Teletek's SBCs will run at 4, 5, 6, or 8MHz and are available with up to 512K of onboard dynamic RAM. The SBC 86/87 also offers an optional 8087 math coprocessor for numeric intensive applications.

Teletek's Systemaster II provides two RS232C serial ports and two Centronics-compatible parallel ports or may be optionally configured to provide a SCSI interface or an IEEE-488 interface to support many laboratory testing and measuring instruments.

Teletek also offers a dual controller board, the HD/CTC, which will control any two ST506-compatible hard disk drives and any QIC-02/QIC-24-compatible cartridge tape drive. This unique design saves hardware cost as well as space in the mainframe.

Teletek's IEEE 696/S-100 boards run under the multi-user, multi-processing operating system TurboDOS by Software 2000, supporting up to 16 users and capable of running PC-DOS application programs. For single-user systems, Teletek is supported by the CP/M operating system.

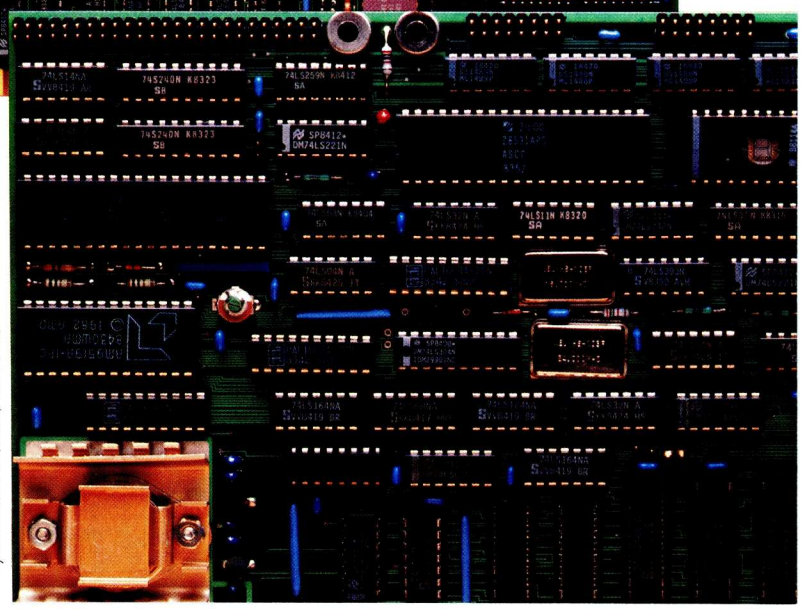


SBC-I

SBC-II

HD/CTC

SBC 86/87



Systemaster II (128K, 6/8 MHz)

In Europe:
Kode Limited
Station Road
Calne, Wiltshire
SN11 OJR England
tel: 0249-813771
telex: 449335

In Brazil:
DANVIC S.A.
R. Conselheiro
Nebias, 1409
01203 São Paulo, Brazil
tel: 221-6033 (PABX)
telex: 1123888 CICP BR

Teletek provides a 30 day evaluation program to qualified customers. For more information, call our Sales Department at 916-920-4600 or write for our information package.

TELETEK

4600 Pell Drive
Sacramento, CA 95638
916-920-4600

For the Advanced Computer User

Micro/Systems Journal™

May/June 1985
Vol. 1 No. 2

SUBSCRIPTION RATES

1 year	$18
2 years	$32
1 year (Canada & Mexico)	$24
2 years (Canada & Mexico)	$44
1 year (other foreign)	$32
2 years (other foreign)	$58

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

ADVERTISING RATES: Available on request. Call (201)522-9347.

CHANGE OF ADDRESS: Please send old label and new address to: Micro/Systems Journal, Box 1192, Mountainside, NJ 07092.

Micro/Systems Journal is published bi-monthly by Libes Inc., 995 Chimney Ridge, Springfield NJ 07081. Application to mail at Second Class Rates is pending at Springfield, New Jersey and additional mailing offices.

POSTMASTER: Please send address changes to Micro/Systems Journal, P.O.Box 1192, Mountainside NJ 07092.

IN THIS ISSUE

FEATURE ARTICLES

PRODUCT PREVIEWS

DEPARTMENTS

AUTHORS: Micro/Systems Journal is always seeking good articles. Please write or call first to see if we are interested in the subject. Please do not send the article unless we ask for it.

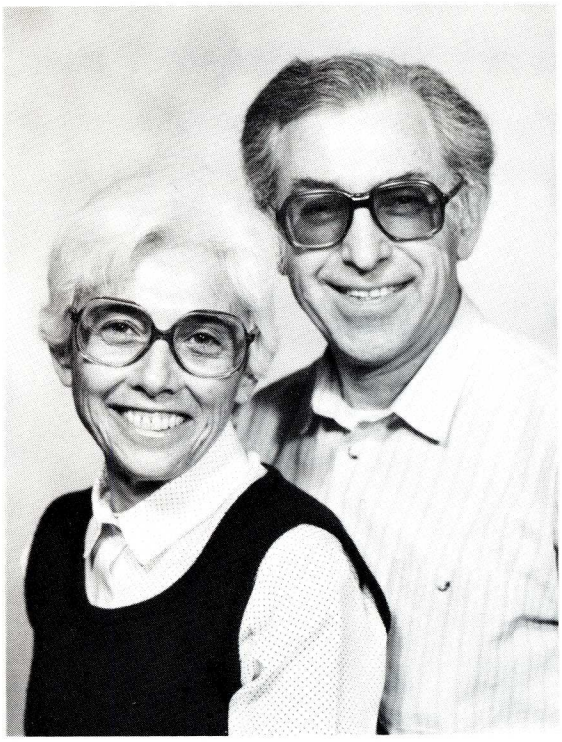
If you are interested in reviewing hardware or software please write telling us your interests, your background and include a sample of your writings.

Send a stamped self-addressed business size envelope for a copy of our Author's Guide.

TRADEMARK ACKNOWLEDGEMENTS

Macintosh	Apple Computer Company
UNIX	AT&T
GEM, CP/M	Digital Research Inc.
Concurrent DOS	Digital Research Inc.
8088, 8086, 80286	Intel Corporation
IBM PC, PC/XT, AT	International Business Machines
MS-DOS, XENIX	Microsoft
GW Basic	Microsoft
68000, 68010,68020	Motorola
dBase II & dBase III	Ashton-Tate

PUBLISHERS PAGE



I would like to take a moment out of a very hectic day to applaud all the *Micro/Systems Journal* subscribers.

For weeks now I have personally processed thousands of subscriptions and have felt the pleasure, excitement and warmth that comes from reading your comments. Almost every subscription contained a comment!

You've expressed hopes, encouragement, and ideas for the future of *Micro/Systems Journal*. Almost everyone wants to be part of it.

I would like to extend a special "thank you" to those who refused to take either of the discounts offered and to those who gave even more than the un-discounted rate, because they wanted so much to help to get this magazine off and running. They felt that this was the least they could do to lend their support.

And, there are those.... many, many of you.... who are receiving magazines from Ziff-Davis you never ordered and even now don't want. You wrote about it in anger and frustration. I wish we could help, but most of you realize that we have no connection with the former publishers of Microsystems magazine. I wrote to many of you suggesting what you might do to get a refund. What I suggested was based on the experiences of those who had already succeeded in getting that refund. Many felt a touch of triumph as they traded in their Ziff-Davis "refund" check for a *Micro/Systems Journal* subscription.

So, our home has again become a publishing house.... our basement a stockroom and mailroom.... and, our ping-pong table once again a production area. The Post Office and U.P.S have dedicated a driver and truck to the cause.

But, if I ever wonder "why did I do it again?", all I have to do is look at Sol (if I can find him) and read some more subscriber comments.

I guess, like Sol, we're all crazy.

Lennie Libes, Co-Publisher

....and, from the Editor

It is with deep regret that I report to you that Ziff-Davis has done it again..... they have closed up yet another magazine. COMPUTERS & ELECTRONICS magazine ceased publication with the April issue. We all remember when C&E was called POPULAR ELECTRONICS. And, in particular we remember the January 1975 issue that carried the article on how to build the Altair Computer, the first S-100 system. PE was a pioneering magazine dedicated to the electronics hobbyist. Les Solomon, the technical editor, and authors, such as Forest Mims, made it a consistently worthwhile publication. I was a loyal subscriber for over 30 years. And, during its last year and a half wrote a regular column for them.

Ziff-Davis has now closed about a half dozen computer magazines, leaving itself only six. One wonders which one will be next.

C&E had a circulation of close to 600,000; the highest in the computer magazine biz. When it was PE, it had a circulation of about 400,000 and was clearly dedicated to electronic hobbyists. But when Z-D refocused the magazine to the computer field, it lost loyal readership (replacing it with expensive transient readers) and lost advertisers (who could no longer afford the highest ad rate in the industry and were not sure of readership interests). Z-D could have admitted its mistake and changed the magazine back to its original focus, reduced its circulation and ad rates and moved it back into the black. Regretfully Z-D decided to take the easy way out and close up a magazine that had a large devoted following.

One wonders..... which magazine will former C&E subscribers now find in their mailboxes?

We have received well over 2,500 personal letters from subscribers wishing us well and offering suggestions as to the future direction of this magazine. We wish we had the room to publish them all, but they would have taken up this entire issue and maybe more. Rather, we have selected a few representative letters and messages and you will find them in the following "Letters" section.

We read every letter we receive and take them to heart, so keep the letters coming.

Micro/Systems Journal Software in Public Domain

All of the software listings published in Micro/Systems Journal has been placed into the public domain. Thus, CP/M software listings will be found in the SIG/M public domain software library and MS/DOS software listings will be found in the PC/Blue public domain software library. I therefore suggest contacting your local computer club to obtain machine readable copies.

If you have problems obtaining any of this software thru public domain channels then you may obtain copies directly from us. To do this send a note indicating the software desired and the disk format together with a check for $10 ($15 outside of North America). Mail to: Microsystems Journal, Box 1192, Mountainside NJ 07092.

Sol Libes
Editor & Co-Publisher

You can still buy quality and dependability at a reasonable price.

That's exactly what we drive home at Viasyn. We offer you a whole line of CompuPro® IEEE 696/S-100 Bus boards. Along with single- and multi-user systems compatible with over 3,000 standard business applications, plus a wide variety of scientific and industrial programs. Each one's a value our competition finds hard to beat.

You can choose from a broad range of CPU boards, memory boards, disk controllers, network and interface boards, plus PC Video, in color or black and white. You can mix or match 8-bit and 16-bit software on the same machine or on different machines. And pick just the power and memory you need. Object: to grow and multiply in capabilities, at optimum cost efficiencies. Without sacrificing quality or dependability.

If that's what you're driving for, write us for our new short-form catalog or simply call our toll-free number.

VIASYN™

The CompuPro People

Where Computers Grow

3506 Breakwater Court, Hayward, CA 94545
Call 800/VIASYN-1. In CA, 800/VIASYN-2. TWX: 510-100-3288 VIASYN CORP

discontinued

dBASE II

(MULTI-USER VERSION)

PCWEEK

FEBRUARY 5, 1985
VOL. 2 NO. 5 $2.95

THE NATIONAL NEWSPAPER OF IBM STANDARD MICROCOMPUTING

Ashton-Tate Pulls 'Multi-User dBASE II' From U.S., Canada Because of Poor Sales

CULVER CITY, CA—Poor sales have forced Ashton-Tate to discontinue the multiuser version of *dBASE II* in the United States and Canada, according to a company spokesman.

The move appears to be a blow to 3Com, makers of the only network on which *Multi-User dBASE II* now runs. The Ashton-Tate spokesman's comments seemed clearly to indicate that *Multi-User dBASE III*, due this spring, will not run on the 3C

NOW, THE RIGHT CHOICE IS EASIER THAN EVER.

DATAFLEX™

THE TRUE MULTI-USER APPLICATION DEVELOPMENT DATA BASE

DATA ACCESS CORPORATION
8525 SW 129 Terrace, Miami, FL 33156-6565 (305) 238-0012
Telex 469021 DATA ACCESS CI

Compatible with MSDOS, PC-DOS, CP/M, CP/M-86, MP/M-86, TurboDOS, Novell Sharenet, PC-Net, Molecular N-Star, Micromation M/Net, Action DPC/OS, OMNINET, IBM PC w/Corvus and OSM Muse.

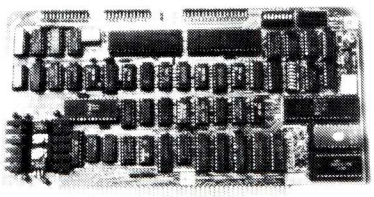
MSDOS is a trademark of Microsoft. CP/M and MP/M are trademarks of Digital Research. DataFlex and FlexKeys are trademarks of Data Access Corp.

dBASE II is a trademark of Ashton-Tate

Quality S-100 Products

FULCRUM: *The S-100 Specialists*

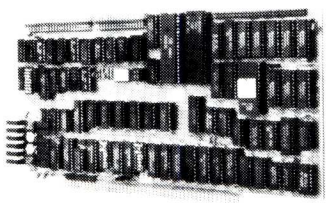
Introducing...
FULCRUM'S NEW
MPUZ CPU



This NEW MPUZ CPU utilizes the Z-80 8MHz uP as a basis for its 8MHz CPU for S-100 systems, and has been carefully designed to meet the requirements of the IEEE - 696 standard. The quality and performance this CPU provides is rarely found in S-100 products, and you can see why...***only $349.***

- ► 4 or 8MHz clock rate
- ► Two RS-232 serial ports
- ► Centronics printer ports
- ► Real time clock with battery back-up
- ► Vectored interrups to any block location in memory
- ► Programmable timer
- ► ROM monitor
- ► Power on Jump
- ► On board wait states
- ► 2K of RAM space
- ► 24-bit extended addressing
- ► Latched Status
- ► Front panel compatibility
- ► MPM support

Best Value In Disk Controllers...
OMNIDISK



Now the FULCRUM OMNIDISK offers S-100 systems users a unique marriage of component compatability and technological innovation. These together produce features not found in any conventional disk controllers made today. See for yourself what tomorrow looks like...***only $349.***

- ► Simultaneous support of both 5¼" and 8" floppy disks and hard disks
- ► Complete 24 bit DMA
- ► Power on boot for 5¼" and 8" floppy and hard disks
- ► Power on boot PROM
- ► On board de-blocking to save RAM space over BIOS
- ► Interfaces with the WD 1001® hard disk controller
- ► Supports 13 devices simulatneously
- ► Full track buffer allows controller to recall entire track
- ► DMA'S at 10 MHz
- ► Supports MS DOS
- ► 10K on board buffer saves two K of TPA

So before you buy another S-100 component, call or write for our *FREE* catalog. And see how your system can benefit from the FULCRUM difference.

*CP/M *2.2 configured for OMNIDISK $60. *Trade mark of Digital Research. FREE U.P.S. ground shipping on prepaid orders. Shipping is added to VISA, M/C, and C.O.D. orders. CA residents, please add sales tax.

FULCRUM COMPUTER PRODUCTS

707/433-0202 459 Allan Court, Healdsburg, CA 95448,

Also in FULCRUM'S Family: OMNIRAM 64K memory board. Serial I/O 2-2 & Video I/O Interface boards, Relay board, I-8080, 8015 and 8035 main frames with 21 slot mother boards, DPA front panel and A/D board.

News, Views & Gossip

by Sol Libes

Random Rumors & Gossip

Look for Borland to shortly introduce version 3.0 of Turbo Pascal, without a doubt the most popular micro version of Pascal. Improvements include much faster compile time, faster execution of compiled code, turtle graphics, and new file I/O system. Also rumored coming from Borland later this year are versions of C, Modula-II, and a spreadsheet to compete with Lotus 1-2-3 ..... Microsoft is expected to shortly announce Version 4 of MSDOS which will, among other things, add a Virtual Memory system to the AT. Expect Microsoft to include a plug-in card to provide the VM hardware support.

I hear that a manufacturer (nameless at this point) plans to introduce a low-cost Z80-based system with ZCPR3 and the ZDOS operating system instead of CP/M-80..... and Spectravideo, Fremont CA, is rumored about to introduce a CP/M-80-based portable, with 3.5'' floppy drive and 25 line x 80 character LCD display for well under $1,000..... and Viasyn (formerly CompuPro, nee Godbout Electronics) is expected, this fall, to release a version of UNIX for its 16032 CPU card.

IBM has signed a contract with MiniScribe for 20Mbyte, 3.5'' hard disk drives. Now which new product could this be for? And there are rumors of a System 36 version of the AT coming with a link to the IBM PC network. Also known to be in the works is a laser printer capable of 20 pages per minute.

There are reports that 256K and 64K RAM chips are already selling for $4 and 80 cents, respectively, on the large volume spot market. Last December, 256K chips were over $8. Predictions are that prices will fall even further this year..... Texas Instruments, Compaq and several other manufacturers are expected to shortly introduce AT-clones. Looks like the AT bus, hardware and operating system may become a defacto standard for 80286-based systems.

MS-DOS on S-100 Systems

More and more S-100 users are upgrading their systems into the 16-bit world with 8088, 8086, 80186 and 80286 CPUs in their systems. Most of these users would like to use the MS/DOS operating system instead of CP/M-86, mainly because of the large software base in existence for MS-DOS. However, there is a problem here. One can buy a copy of MS-DOS for the IBM-PC and implementations for most of the PC clones. However, because of hardware differences they will not even boot.

Computer House, Woodacre CA, some time ago developed a version for CompuPro systems. Now, many of the S-100 manufacturers are tailoring their hardware systems to support standard PC-DOS. For example, Tarbell Electronics, Carson CA has introduced a new 80286-based CPU card that will boot up a standard IBM-PC DOS disk without any modifications required. And, if the system contains a PC-compatible video card (Lomas Data already has such an S-100 card and Viasyn is expected to start shipping theirs soon) then one can run things like Lotus 1-2-3 with lightening speed. Even without the video card programs such as WordStar, Multiplan and dbase-II can be run. Further, there is a great likelihood that these systems will be able to boot and run other operating systems written for the PC and AT (e.g. XENIX and PICK).

Companies such as Lomas Data and Seattle Computer Products already sell versions of MS-DOS to run with their hardware. And, if there is a reader who would like to write an article on how to modify PC-DOS to run with S-100 16-bit CPUs whose manufacturers do not supply MS-DOS please contact me. We are looking to publish such an article.

PC-II Where Are You?

Speculation is that IBM has held back introducing the new version of the PC because they have warehouses full of PCs and XTs which are moving very slowly. Only when inventory is cleared out is IBM expected to release the PC-II. The most important feature of the PC-II is expected to be its design for high volume automated mass production to reduce production costs.

The unit is expected to use the 80186 or 80286 microprocessor with restricted performance (e.g. less memory addressing capability) so as not to impact sales of the AT. Further, IBM is expected to retain the 5.25'' disk format and offer a 3.5'' drive as an option. IBM has done extensive market research and concluded that its cutomers, with well over 3 million PCs already in use, would be very upset if IBM abandoned the 5'' drive. Hence, IBM is expected to provide a bridge to the 3.5'', 1.6Mbyte drive, which they are expected to use in their new portable (expected this summer) and other new systems coming later this year and next. IBM would like to get the desk footprint of their new machine to be significantly smaller than that of the PC.

In the meantime, IBM will halt production of the PCjr, only 16 months after it was introduced. It marks IBM's most visible failure in the personal computer marketplace. Actually this is IBM's third failure in the personal computer marketplace as the two predecessors to the PC (the 5100 and 5110) also flopped and IBM has not done too well with the PC-portable. However, their success with the PC and AT have much more than compensated for their bombs. Also, the XT (particularly since the introduction of the AT) has been a poor sales performer.

IBM invested close to $50 million promoting the PCjr and cut the price drastically for the Christmas selling season, but since the first of the year sales have been nil as IBM restored the regular price for the machine. Earlier IBM replaced several thousand keyboards when reviewers, dealers and customers complained. In all, IBM sold about 270,000 jrs last year, with 75% of the systems sold in the last quarter when the complete systems price dropped to under $1000 list, with a street price between $800-900, including a color monitor and software. IBM's abandonment, leaves Apple, Sanyo, Atari and Commodore as the major competitors in the home computer market. However, Sinclair and several Japanese and far east suppliers are expected to cautiously enter the chaotic U.S. market this year. Sanyo,

who introduced an MS-DOS machine with limited PC capability, has over the last two years established a strong dealer network and a large and loyal user base. Recently, they improved the PC-compatibility of the machine and its performance. The price for a basic system is well under $1,000 and is heavily discounted by most dealers, making it very attractive as a home system.

The Check Is in the Mail

Imagine a microcomputer manufacturer shipping $6 million in merchandise to a retailer and not being paid. If it had happened to most any other manufacturer it probably would have forced the company into bankruptcy. But not IBM... it takes it in stride.

IBM filed suit against Computermate, a California computer retailer, to which it shipped 400 XTs and other stuff totalling $6 million. The retailer sent IBM a check which bounced and then a second check which also bounced. Finally, IBM filed suit against the retailer.

Newsletters of Note

"null-Babel/CBNnews" is a newsletter for CBasic users. Published monthly a subscription is $18/yr US and Canada and $28/yr foreign. Write to: Ric Allan, Box 40690, Cincinnati OH 45240, or call: (513)851-4774, after 7PM EST.

"Z-NEWS" is a monthly newsletter supporting users of ZCPR3 and the new ZDOS disk operating system. It is published by Echelon Inc., 101 First Street, Los Altos CA 94022, telephone (415)948-3820. Echelon also supports a bulletin board system for ZCPR3 and ZDOS users. Called "Z-Node" it is at (415)489-9005.

Public Domain Software News

The C User's Group's public domain C software library now consists of 44 volumes of software which are available in 69 different 5" and 8" disk formats. The group also publishes an infrequent newsletter. For information on the software library and newsletter write to: The C User's Group, 415 E Euclid, Box 97, McPherson KS 67460 or call (316)241-1065.

Random Bits

Bill Gates, Microsoft Corp. chairman, announced at a recent independent software vendor gathering that they are projecting that over 400,000 copies of Xenix/286 will be sold. If this comes to pass Xenix will be *the* most popular operating system for small multi-user computer systems and could become the defacto standard for UNIX.

there is mail.....

Dear Sol:

I must be crazy too! Sign me up for two years. Enclosed is the Ziff-Davis check that I received as a refund on my old Microsystems subscription. I have endorsed it over to *Micro/Systems Journal*.

I had a bit of a hassle getting the refund. I got my refund by writing to: Shane G. Boel, Circulation Manager, PC Tech Journal, Box 2968, Boulder CO 80322. I told him that I did not want a magazine I did not subscribe to and wanted a refund on the remaining portion of by sub. I had to wait about three months but they finally came through.

Good Luck.

David Robinson
Bismark North Dakota

Dear Sol:

I have the following suggestions:

a) Aim at the frontiers... Z8000, National 32016/32032 and Intel 80286/386. Urge the promotion of a standard for 32-bit processors on the S-100 bus.

b) All code for MS-DOS and CP/M-86 should be *generic*... should only use standard system calls and not depend on IBM-PC hardware.

c) I suggest that Turbo Pascal code be the standard.

Robert Hanrahan
Gainsville Florida

Dear Sol:

Thank you so much for doing it again. Of the many computer magazines I have subscribed to over the years, Microsystems was my favorite. Keep the good stuff coming on CP/M, C, UNIX and the hell with IBM.

Ed Roberg Jr
Clearlake CA

Folks,

Please enter me as a subscriber to Micro/Systems Journal. Delighted to hear it exists. This time... PLEASE, PLEASE, PROMISE... not to sell it to the City Slickers, ok?

Dave Cortesi
Palo Alto CA

Sol:

I was a subscriber to Microsystems from Vol 1, No 1. I would like to subscribe to your new magazine, however Ziff-Davis has not yet refunded by money, even after several letters. When they do I will probably subscribe.

I owned one of the first computer stores so really have been active for a while and enjoyed Microsystems! Looks like the New York "bean counters" got to you.

Good luck!

Chuck Suit
Silver Spring Maryland

Sol:

Don't enter my subscription!

I do not want to start another subscription and have it turned into a worthless magazine!

Forrest I Gompf
San Diego California

Dear Sol:

Although I still use CP/M I have a Zenith 160 PC compatible and so would like to see PC compatible articles also... especially on MS/DOS, PC/Blue and C-User Group.

Cole Ellsworth
Garden Grove California

Dear Mr. Libes:

I was very much disappointed when I was informed by Ziff-Davis that Microsystems ceased publication. My "first" subscribed issue of Microsystems happened to be the "last" issue. I was very resentful when I was offered an IBM oriented magazine in lieu of my sub to Microsystems.

I am glad that you and your wife are back to keep us hackers happy. The best feature of "Microsystems" has been, to me, that it has managed to keep its eyes to the future computer trends, satisfying at the same time hackers' need to work with micros they have now. This unique blend of helping us with micros we have and also assisting us to see what's in the future will be found in your new journal, I hope! Welcome back among the hackers who need you.

Paul Naitoh
San Diego California

Dear Sol:

Welcome back! My life "also" doesn't seem the same without you and Microsystems for evening comfort and knowledge expansion. Even though there is a plethora of computer magazines on the racks, your new magazine is needed.

I have a problem and would appreciate some help from you readers.

Letters to M/S-J

We welcome your letters with comments, compliments, criticism and suggestions. We read them all and publish the most noteworthy, even if they are critical of us. We do not have the staff to answer all letters personally. And all letters become the property of M/S-J and may be subject to editing. Further, we do not print letters that do not include a name and address.

Please send your letters to: Micro/Systems Journal, Box 1192, Mountainside NJ 07092.

I am a one-man company specializing in the construction of vertical software for CP/M, MS-DOS and VAX/VMS software. Currently, I am in a bit of bind moving a program from CP/M to MS-DOS. My system consists of interacting programs written in Digital Research's CB80 which I can also compile under CB86. I have an S-100 system from U.S. Micro Sales, made by XOR Data Systems, which has two 8" DSDD drives and a controller which handles both 8" and 5-1/4" drives. I want to know is it possible to format and copy to MS-DOS 5-1/4" disks from my 8" drives while working under CP/M?

Melvin W Smith,
SYNTEL Data Systems Ltd
Ste. #1111,
10080 Jasper Ave
Edmonton, Alberta,
Canada T5J 1V9

Whenever I want to move files between systems with incompatible disk formats I will either use a null modem interface between the two systems and a data transfer or communications program to move the data. On occasion I have even uploaded the file from one system to the other via modem and the telephone line. This works very nicely for Basic and ASCII files.

Dear Mr. Libes

I want to wish you well in your new endeavor and to tell you that the micro world will get along just fine without the old Microsystems. Let me tell you why I will not be subscribing. I think you are living in the past. Way too much stuff on CP/M, S-100 bus and such. I have been in the PC/MS-DOS camp for 2 years now and am very glad I swore off (and at!) CP/M. You might be able to fool some potential subscribers into thinking you can teach an old dog new tricks, but not me.

If the early issues indicate that I am wrong and contain numerous current original articles on and about non-CP/

M systems, I will admit I am wrong.

Wandal W Winn
Anchorage Alaska

Dear Sol:

Are there going to be any MS-DOS/S-100 articles?

Tony Price
Burbank California

Sirs:

I would very much like to see an article or two addressing the CompuPro 8/16 hardware and software. I am also interested in the Macrotech MI-285 Z-80H/80286 CPU replacement for the Godbout one. Are there any pitfalls or contraints? Will an S-100 color video board be available to run PC graphics? Will the newly released DRI Concurrent-DOS run with the Macrotech CPU?

Dan Nelson
Phoenix Arizona

We expect to be addressing these questions in articles now being written.

Sol:

Glad to see you back. The last issue was the best. Hope you can keep it up.

R. C. Howard

Dear Sol:

Are you going to continue to write for Micro-Cornucopia?

M. H. Noble
Pawcatuck CT

Micro-Cornucopia is a fine magazine and I tried to interest Dave Thompson, the publisher, in publishing this magazine with me as editor. Actually, I tried to interest several publishers with no luck and hence I am back in the magazine publishing biz again when all I wanted was to be an editor.

Micro/Systems Journal is taking up all my time and forced me to drop all of my other projects including writing for Micro-C.

Sol:

Best of luck to a first class masochist! You deserve the best but probably won't get it. Ziff-Davis will probably make life miserable for you the way IBM does to small companies. Long live the underdog!

Art Hurst
Woodland Hills CA

Dear Sol:

. . . hope you will include articles on C programming language as well as the UNIX operating system.

Charles D'Englere
Decatur GA

Dear Sol:

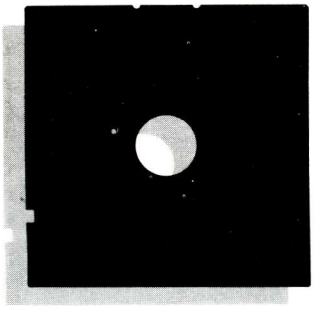
I ought to be mad at you, but I cannot hold a grudge. I had cancelled all my subscriptions to computer magazines except yours. Then you sold out to Ziff-Davis and they closed Microsystems. I was sorry to see you old timers pass from the scene.

Then I got your letter and I was really excited about something good to read again. I wish you all the luck. . . you will need it to make it.

J. E. Murray
Columbia SC

We did not sell out to Ziff-Davis. We sold the magazine to Dave Ahl of Creative Computing. When Z-D bought the CC operation they got their hands on Microsystems.

The former President of the Z-D computer magazine group supported Microsystems and tried to help it grow. He allowed me to hire Chris Terry as technical editor and to go monthly. Many of our best issues were done during this period. When he was fired the new President took one look at the bottom line dollars and decided that, although Microsystems was profitable, more money could be made by investing in other areas.



In the SIG/M Public Domain

by Stephen M. Leon

Some months back SIG/M received a proposed contribution of a disk called *Death*. A battle with aliens from outer space? A duel to the end of the knights of old? No, it was a checklist of things to do before the inevitable happened!

The author had put a lot of time and effort into writing it. Some parts made sense, and some of it could have caused problems if one did not consult with an attorney. It would not have been acceptable for the library in any event. However, what killed "Death" was the fact that the author requested the user send him a small contribution. Death might be inevitable, but our author felt it should not be free.

In talking to the authors and proponents of Freeware or Shareware one is told that they cannot afford to market the program. They want to use the worldwide facilities of the public domain distribution network for nothing — and then they write us nasty letters when we say no! When you write the Great American novel and the publishers turn you down, your choice is either vanity press or forget it. How many commercials does CBS air for nothing because the sponsor can't pay? The Borland people have clearly demonstrated how a good product at a reasonable price will sell, even when marketed on a shoe string budget. The Freeware backers contend that since the option to pay or not is the users, it is beneficial to the users to distribute it in the public domain network. Talk to them and they will tell you that without distribution on the bulletin boards and in the public domain libraries, the authors would not produce these works. If any of you have been to downtown New York lately during a weekday lunch hour, you know that you can barely walk on the sidewalk because of all the street peddlers. The street peddlers are taking business from legitimate stores. The stores pay taxes, are around when you need to exchange something, etc. Same thing with some of the Freeware peddlers. A small ad in Computer Shopper or one of the magazines should not be too much of a burden. To most of them Freeware is a moonlighting project, in many cases developed on the bosses time. What about sales tax? What about income tax? One may need a million dollar budget to hit the top of the software charts, but what Freeware developer has the overhead of Ashton-Tate?

We certainly should do everything possible to encourage the development of good, inexpensive software. Freeware doesn't encourage it. It lets someone test market for free. Rather, let that person make contributions to the public domain without restriction, as for example, Rich Conn. When Rich offers a product commercially his name and reputation made in the field of public domain software is enough to sell the product.

Speaking of taxes, one can kick around the question as to whether or not a tax exempt organization such as SIG/M could actively distribute Freeware and still retain its tax status. In any event, the anti-freeware policy of SIG/M has not produced a shortage of good software donations to the dismay of the advocates of Freeware.

We have added to the library (SIG/M Volumes 218, 219, and 220) Ron Fowler's MEX. MEX represents a significant improvement in the field of CP/M 80 modem programs. MEX brings Ward Christensen's MODEM 7 up one generation. The Fowler program is good, but not without controversy. In putting it into the library we did so knowing that Ron did not provide the source code, which code is normally routinely provided with public domain programs. Freeware too does not provide the source code, but for a different reason. Fowler did it to protect the program's code from the tinkerers who put out 43 updates in two weeks. Ron went commercial with the 16 bit version of the program — but he did it as a commercial operation and not with hat in hand. We might add that the price he is selling the 16 bit version is not much different from the 'Give Me' sign that comes up everytime one runs the Freeware.

Steve Leon is the SIG/M Disk Editor. In other words, he is the person who assembles, compiles and edits all the of the SIG/M public domain software disks. Thus, he speaks with the greatest authority as to what is going on in the SIG/M public domain software area.

Speaking of 16 bit, Bill Earnest has taken RESOURCE, one of the better disassemblers, and turned it into a 16 bit program. RES86 on SIG/M Volume 223 is a handy tool for CP/M 86 users. Also on 223 are some NEC APC utilities and a program to set the function keys on the Wyse 50.

SIG/M Volume 221 contains programs for dBASEII users, including a complete inventory program. Volume 222 has some utilities updates. Included is an improved DBL. DBL allows a dot matrix printers to print double columns of type for newsletters and the like. Quite a handy utility for your Epson or Okidata printer.

SIG/M Volumes are available on 8" SS SD Disks for $6.00 each ($9.00 foreign) directly from SIG/M, Box 97, Iselin, NJ 08830. Printed catalogs are $3.00 each ($4.00 foreign). Disks in a variety of formats may be obtained through the world wide SIG/M distribution network. The distribution list is included with the printed catalog. A disk version of the catalog (Volume 00) is available for $6.00.

SIG/M is a non-profit group operated by non-paid volunteers and is a sub-group of the Amateur Computer Group of New Jersey, Inc.

The following are the contents of the most recent SIG/M volume releases:

Vol 218 MEX - Improved Modem for CP/M
by Ron Fowler (Volume 1 of 3)

MEX112	.OBJ	MEX main program
MEX	.HQP	On-line users manual
MEX10	.DQC	Documentation
MEXSUM	.DQC	/
MEXCON	.SUB	Submit file
MLOAD	.OBJ	Overlay install tool

READ	.MQ	Quick overview
SET	.MQX	Changes MEX settings
TEL	.PHN	Sample phone dir.
MXM-CD10	.AQM	Concord Data System
MXM-PC10	.AQM	Popcon
MXM-UD10	.AQM	UDS modem
MXM-US13	.AQM	US Robotics modem
MXO-AD11	.AQM	Advanced Digital
MXO-AL10	.AQM	Altos
MXO-PM22	.AQM	PMMI
MXO-SM14	.AQM	Smartmodem

Vol 219 MEX - Improved Modem for CP/M by Ron Fowler (Volume 2 of 3)

MXO-APCC	.AQM	Apple II overlay
MXO-CT10	.AQM	CT Companion
MXO-DB10	.AQM	Dynabyte
MXO-DT10	.AQM	Datec
MXO-DV10	.AQM	Davidge
MXO-EP12	.AQM	Epson QX 10
MXO-GB11	.AQM	CompuPro
MXO-H812	.AQM	Heath H-89
MXO-II12	.AQM	Intersystems
MXO-IM10	.AQM	Info-Mate
MXO-KP41	.AQM	Kaypro
MXO-LO15	.AQM	Lobo Max 80
MXO-MD11	.AQM	Morrow MD 3A
MXO-MG10	.AQM	Mega SC
MXO-MR10	.AQM	Morrow Micro Dec.
MXO-NE88	.AQM	NEC PC-8801
MXO-NS11	.AQM	North Star Horizon

Vol 220 MEX - Improved Modem for CP/M by Ron Fowler (Volume 3 of 3)

MXO-OA11	.AQM	Otrona Attache
MXO-OC10	.AQM	Osborne 1 w/COMM-PAC
MXO-OS22	.AQM	Obsorne
MXO-OX11	.AQM	Osborne Executive
MXO-P1-1	.AQM	PCM Micromate
MXO-R+10	.AQM	RS Mod 4 with CP/M+
MXO-R211	.AQM	Radio Shack Mod 2
MXO-RS13	.AQM	RS Mod 4 w/Montezuma
MXO-RV13	.AQM	Racal- Vadic
MXO-SB12	.AQM	Superbrain
MXO-SCAT	.AQM	Smartcat
MXO-SX10.	AQM	Starplex
MXO-SY21	.AQM	Sanyo MBC-100
MXO-TD30	.AQM	Turbodos
MXO-TV11	.AQM	Televideo TS-802
MXO-VP10	.AQM	Ventel Plus
MXO-VTL1	.AQM	Ventel w/Otrona Att.
MXO-XE12	.AQM	Xerox 820
MXO-XR10	.AQM	XOR S-100-4

Vol 221 dBASEII(tm) Inventory System, Decoder, Patch and MP/M Info.

DBASEMPM	.LBR	MP/M setup programs
DBSOURCE	.LBR	Decodes run-time
INVNTORY	.LBR	Inventory system
NEWBASE5	.LBR	dBASE patches

Vol 222 Updates to Handy Utilities & Misc. Programs

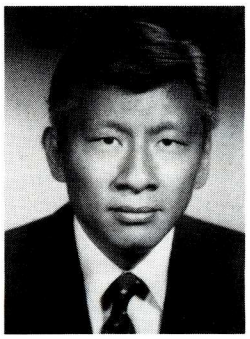
AREACD13	.LBR	New areacodes
DASM8080	.LBR	8080 disassembler
DBL003	.LBR	Multi column print
DIRREP	.LBR	Repairs directory
LU310	.LBR	Updated library util
NULU10	.LBR	A new approach to LU

Vol 223 RESOURCE Disassembler for CP/M 86 Misc. Programs

DBIIDATE	.PRG	NEC APC dBASE date
DO	.LBR	Extended SUBMIT
FREEBASE	.LBR	Data base program
MEM	.LBR	Creates RAM disk
PDABACUS	.CMD	Sets colors - NEC APC
PDRAMDSK	.AQ6	RAM disk for NEC APC
PDRAMDSK	.CMD	/
RES86	.LBR	Disassembler-CP-M 86
SPLITTER	.LBR	Segments files
TLABEL86	.LBR	Prints labels-CP-M 86
WYSEFK21	.AQM	Wyse 50 function keys.

PC/Blue Report

by Hank Kee



The trend in *public domain* MS/DOS software distribution is increasingly through the *user-support* concept. Most authors are requesting a *nominal* donation. I have found there is no correlation between the requested donation and the quality of the software offered. Further, most of these programs are only available in object code form.

The authors are using the public domain as a way to enter the commercial software market without the inherent expense of media advertising. This is not the ideal way to build a public domain software library but it seems to be the common method of contribution used by most authors in the IBM PC environment.

A very popular commercial software product today is *Sidekick*. This has spawned many other similar programs which sit quietly in resident memory ready to be invoked at the user's request. These programs are multifunction in nature. They are not full-featured electronic spreadsheets nor word processors nor communication systems. These programs on the other hand offer calculator emulation, notepad reference, and system utility oriented support. A very neat one in public domain is *PC-Window*. This program features: a) alarm clock and split timer, b) scratchpad ASCII file read and write, and c) hex-decimal conversion.

This program does not have the completeness of Sidekick but it certainly has the basics for programmer support. PC-Window is most useful in reminding the overly intent to keep track of his or her time on the machine. One can be so immersed in using the system that one loses track of time. There is more to life than just poking around on the machine.

These resident programs are fairly complete in functionality. But, a word of caution is necessary. These programs work by trapping interrupts. If you put enough of these utilities into resident memory, you will slow down your overall processing. Another caution is that they may not work together. Some of the programs for normal everyday use become resident at load time.

Home requirements are very different from the office workplace. As the PC matures, the focus of attention in software development turns to personal use support systems. We have in the recent releases of the PC/Blue library received some very fine examples of such programs, such as: *Daily Diary* — a time and manager program, *PC/Calculator* — calculator emulation, and *Checkbook Distribution* — a check balancing program.

There are six new releases in the PC/Blue library bringing the total number of volumes to 116. Here are the contents of the new releases.

Volume 111
- Miscellaneous Utilities
- ASCII import into LOTUS
- blank monitor screen after wait
- encryption program
- Epson to Gemini code conversion
- file function queries
- Hebrew graphics extension
- HELP utilities
- one key control of multiple keys
- color graphics command attributes
- stack last ten keyboard commands
- logical reset of memory switches
- Epson MX80 options

Volume 112
- Miscellaneous Utilities
- amortization program
- area code finder
- BASICA menu generator
- calendar generator
- dBASE menu generator
- redefine function keys
- resident keyboard/clock status popup
- MS/PC-DOS file utilities
- sorted directory displays
- global time/date stamp

Volume 113
- Time and Money v1.1B
- A Simple Financial Monitoring System

Editor's Note: Hank Kee is the librarian for the PC/Blue public domain software library. He is the person who collects, assembles, and checks all the software issued by PC/Blue and then compiles and edits them into the released volumes.

Volume 114
- PC-Talk III (assembler version)
- QModem (compatible with 103A)
- Daily Diary v1.0

Volume 115
- ScreenWrite Formatting Program
- PC/Calculator v1.0

Volume 116
- Genealogy (dBASE)
- Church Management System (dBASE)
- Checkbook Management (dBASE)
- Checkbook Distribution

The PC/Blue public domain software library is distributed on double-sided double-density disks formatted to be PC-DOS 1.10 and higher compatible. Clubs wishing to distribute copies of the library to their club members should contact Bob Todd to make arrangements to receive new releases automatically as they are issued. Bob is also the regional distributor of the SIG/M library.

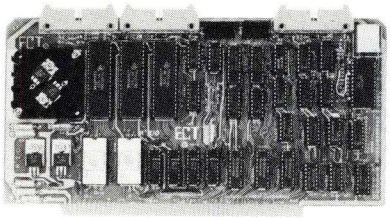
Bob Todd, SIG/M-PC/BLUE Distribution
Amateur Computer Group of NJ
Box 97
Iselin, NJ 08810

The New York Amateur Computer Club handles submittals and individual distribution of PC/Blue disks and catalogs. The disks are $7 domestic and $9 foreign. A catalog describing the contents of each volume as well as topical cross-reference is $5. Foreign orders must be drawn on a US bank.

New York Amateur Computer Club
Box 106
Church Street Station
New York NY 10008

CUSTOM PRODUCTS

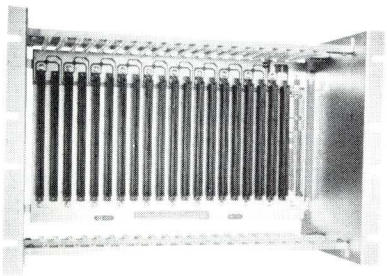
DESIGN • LAYOUT MANUFACTURING



R ²I/O

ROM/RAM & I/O

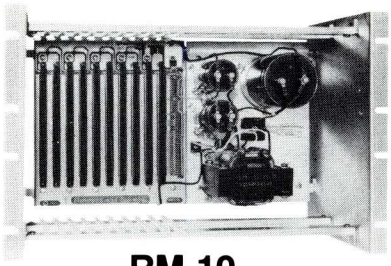
S-100 PRODUCTS



ECT-100-F

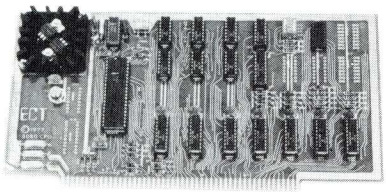
RACKMOUNT CARD CAGES

ECT™



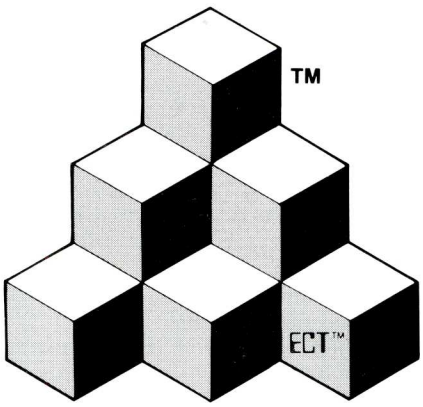
RM-10

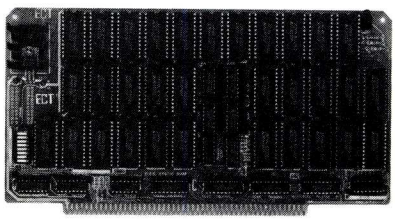
CARD CAGE & POWER SUPPLY



8080 CPU

CENTRAL PROCESSING UNITS



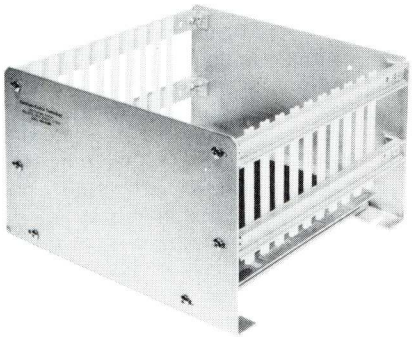


64K RAM

FULLY STATIC MEMORY

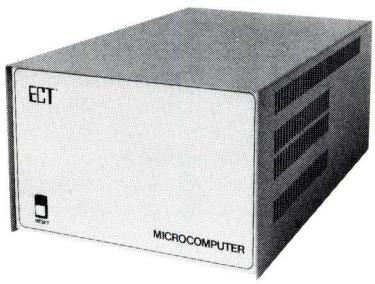
BUILDING BLOCKS FOR MICROCOMPUTER SYSTEMS, DEDICATED CONTROLLERS AND TEST EQUIPMENT

CARD CAGES, POWER SUPPLIES MAINFRAMES, CPU'S, MEMORY I/O, OEM VARIATIONS



CCMB-10-F MIN

6,10 OR 20 SLOT CARD CAGES



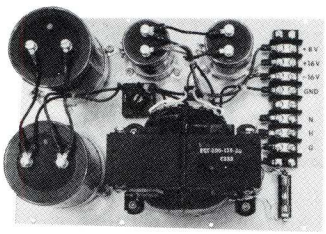
TT-10

TABLE TOP MAINFRAMES

ELECTRONIC CONTROL TECHNOLOGY, INC.

10 Cottage St., Berkeley Heights, NJ 07922 (201) 464-8086

SPECIALIZING IN QUALITY MICRO COMPUTER HARDWARE



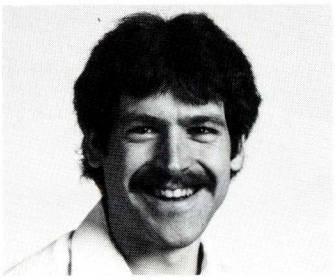
PS-30 A

POWER SUPPLIES

MULTIBUS® PRODUCTS

MULTIBUS IS A TRADEMARK OF INTEL CORP.

The C Forum



by Don Libes

This month, I will discuss building a translation program using the C language that is in the style of a UNIX filter. I mean it to be educational rather than practical, so I have forsaken completeness and instead concentrated on ideas and program readability. My intent is that you should be able take what I've given you here and build on it.

My example will be a program to "decipher" WordStar document files. It's been a while since I've used WS, but for a long time it was my primary tool for word processing. More and more, I find myself using other editors and text processors. Some of the reasons are:

1) WS doesn't support high quality devices such as laser printers.
2) WS doesn't support bitmapped screens.
3) WS isn't as sophisticated as newer editors and text processors.
4) WS isn't integrated with other programs that use text files.

WS's lack of integration is one of its more annoying traits. If you're like me, you've probably typed in a good many documents through WS. Now, say we want to use a WS-text file as input to another program or even another computer system to be integrated into a non-WS document. We've got problems.

If you've ever printed a raw WS document file on your screen (without going through WS), you will have already noticed that there is more in the file than just the text. If you haven't, try it. You will notice that most of the text appears, but some of the characters are wrong and there are a lot of non-printable characters embedded in the file.

All the information is there, but WS uses the high-order bit in the byte for information as well as including extra bytes for print control characters. Unfortunately, MicroPro will not give out the format of their WS document files, but don't let that dissuade you. It's not that difficult to decipher.

Due to space limitations, I simply can't cover all the goodies in WS. I've restricted myself to the things you are most likely to see, however, and following my lead, it should not be hard to extend what I've given you.

Writing a translation program in C

If you want to probe your own files, try loading them into the debugger and dumping the file so that you can see the byte codes alongside the ASCII characters. After some studying it should become clear. This is what I found:

1. Some characters have their most significant bit (MSB) on. This is used to denote several things (see below).
2. Lines are terminated with cr-lf sequences.
3. Dot commands (e.g. ".op") appear in the file verbatim.
4. Blanks, with the MSB on, are soft, meaning they have been added automatically by WS to pad out the line.
5. Hyphens, with the MSB on, are soft, having been added by a rejustify operation (although potentially with user guidance).
6. Linefeeds, with the MSB on, are soft, having been added automatically. Hard linefeeds denote the end of a paragraph.
7. WS print control characters (e.g. ^B) appear in the file verbatim.
8. Tabs do not appear in the file, but are stored as blanks.
9. All other characters appear as themselves in the file. Characters at the ends of words in paragraphs that have been justified have their MSB on.

Assuming that we want to transfer this file to another computer system, we must change the file so that it looks like an ordinary text file. This means that we have to do things like get rid of the high-order bits and remove the WS-dependent directives like the dot commands and the print control characters.

Accompanying this column is a program I've written that handles the basic problems of "unsofting" a WS document file. Most of the more esoteric features aren't handled. However, given the start, you should be able to easily add the code to handle any additional conditions you find really necessary.

You can also customize it to your application. With some simple changes it could be used to generate troff files, for example. The conventions I have chosen to use actually mimic "wsconv", a public domain program from the PC/BLUE software library (distributed without source, unfortunately), author unknown.

I'll briefly go through some of the more interesting parts of the program. I will refer to source lines by the numbers running down the left hand side of the program listing.

The program is written in the style of a typical UNIX filter. It copies its input to its output with appropriate modifications. Internally, the program sits in a loop, reading one character at a time. Based on the character attributes and the current state, the character is printed out and/or the state is changed.

Lines 13-17 define and use #CONTROL. This macro generates control character values given the corresponding letter.

Lines 19-26 define the conventions for handling WS print control characters. For example, an underlined string will print out as "<_string_>". We actually use this convention when sending files to our typesetter.

Line 28: Anything declared as "boolean" should only have the value TRUE or FALSE. This is purely a semantic interpretation since "boolean" is typedef'ed to "int".

Lines 33-46 define several state variables that will be used to make decisions along with the next character.

Lines 53-56: Our first problem is turning off the 8th bit on all characters. This is done by #toascii, defined in <types.h>. To remember that the bit was on, we use msb_was_set.

Lines 57-61: WS terminates lines with a crlf. Assuming our system terminates lines with a nl (like a UNIX system), we can just disregard the cr. If a hyphen added by a justify operation is encountered, we attempt to remove it by delaying line wrap until we see the real end of the current word.

Lines 62-64: For lack of space, I have chosen to ignore handling dot commands. They can be handled here without any problems.

Lines 65-75: Spaces come in two flavors - soft and hard. Hard ones are

```
1   #include <stdio.h>
2   #include <ctype.h>

4   #define TRUE          1
5   #define FALSE         0

7   #define NL            '\n'
8   #define CR            '\r'
9   #define SOFT_HYPHEN   0x1f    /* inserted to break words at eol */

11  #define TABSIZE       8

13  #define CONTROL(x)    ((x)-'a')
14  #define BOLD          (CONTROL('b'))
15  #define SUPERSCRIPT   (CONTROL('t'))
16  #define SUBSCRIPT     (CONTROL('v'))
17  #define UNDERSCORE    (CONTROL('s'))

19  #define BOLD_BEGIN            "<<<"
20  #define BOLD_END              ">>>"
21  #define UNDERSCORE_BEGIN      "<_"
22  #define UNDERSCORE_END        "_>"
23  #define SUBSCRIPT_BEGIN       "<<"
24  #define SUBSCRIPT_END         ">>"
25  #define SUPERSCRIPT_BEGIN     "<^"
26  #define SUPERSCRIPT_END       "^>"

28  typedef int boolean;

30  int column;           /* column to output next character */
31  int c;                /* last character read */

33  boolean superscript = FALSE;/* if superscripting */
34  boolean subscript = FALSE;  /* if subscripting */
35  boolean bold = FALSE;       /* if emboldening */
36  boolean underscore = FALSE; /* if underscoring */
37  boolean msb_was_set;        /* if char most significant bit set */
38  boolean soft_space = FALSE; /* if between words and have seen */
39                              /* soft space, but no other spaces */
40  boolean soft_hyphen = FALSE;/* if in the middle of a word that */
41                              /* was hyphenated by ws */
42  boolean dotcmd = FALSE;     /* if in the middle of a dot cmd */
43  boolean wrap_soon = FALSE;  /* if should wrap as soon as we get */
44                              /* to the end of the current word */

46  int spacerun = 0;           /* number of spaces seen in a row */

48  main() {
49      while (TRUE) {
50          /* look at single characters */
51          msb_was_set = FALSE;
52          if (EOF == (c = getchar())) break;
53          if (!isascii(c)) {  /* is most sig. bit set? */
54              c = toascii(c); /* turn of most sig. bit */
55              msb_was_set = TRUE;
56          }
57          if (c == CR) continue; /* always followed by an NL */
58          if (c == NL) {
59              if (soft_hyphen) {
60                  wrap_soon = TRUE;
61              } else newline();
62          } else if (dotcmd) {
63              /* throw away chars if we are in a dot command */
64              continue;
65          } else if (c == ' ') {
66              /* ignore blanks with msb set - they are soft */
67              if (!msb_was_set) {     /* a real space */
68                  soft_space = FALSE;
69                  spacerun++;
70                  column++;
71                  if (wrap_soon) { /* wrap now! */
72                      newline();
73                      wrap_soon = FALSE;
74                  }
75              } else soft_space = TRUE;
76          } else if (c == SOFT_HYPHEN) {
77              /* ignore hyphens with msb set - they are soft */
78              soft_hyphen = TRUE;
79          } else if (c == '.' && column == 0) {
80              /* text processing directive */
81              dotcmd = TRUE;
82          } else if (iswscntrl(c)) {
83              /* ignore wordstar print control characters */
84              /* e.g. ^S (underscore), ^B (bold) */
85              wscntrl(c);
86          } else if (iscntrl(c)) {
87              /* unknown control character - ignore */
88              continue;
89          } else { /* normal character */
90              /* if we encountered a soft space, stick in at */
```

C Forum Continued from Page 16

real, but soft ones have been added by a justification operation. Tabs are denoted by hard spaces, so we'll try to undo that too.

Line 79 recognizes the start of a dot command.

Lines 82-85 handle WS print control characters. wscntrl() does everything. iswscntrl() just recognizes the characters.

Lines 86-88 remove all the other ws control characters we are too lazy to correctly handle.

Lines 89-104: If the character hasn't been handled at this point, its got to be text, which is just passed through. If we're at the beginning of a word, we also perform the translation of spaces back to tabs here by calling space_out().

Lines 108-130 define space_out(). This routine takes a source and destination column, and prints out the least number of tabs and spaces to move the cursor to the destination column.

Lines 152-180 define wscntrl(). This translates control character directives into printable versions.

I encourage readers to write to me about topics or problems that you want to know about. I want this column to reader driven. Write to me care of Micro/Systems Journal, Box 1192, Mountainside NJ 07092.

```
91              /* least one space */
92              if (soft_space) {
93                  spacerun = 1;
94                  column++;
95                  soft_space = FALSE;
96              }
97              if (spacerun) { /* beginning of word */
98                  /* calculate tabs and blanks to lay down */
99          space_out(column-spacerun,column);
100                 spacerun = 0;
101             }
102             putchar(c);
103             column++;
104         }
105     }
106 }

108 /* print least number of spaces & tabs to move from "oldpos" */
109 /* to "newpos" */
110 space_out(oldpos,curpos)
111 int oldpos;     /* old position */
112 int newpos;     /* new position */
113 {
114     int spaces, tabs;   /* number of spaces & tabs to print */
115     int i;

117     if (oldpos >= newpos) return;     /* no space in between */

119     /* first calculate tabs */
120     tabs = newpos/TABSIZE - oldpos/TABSIZE;

122     /* now calulate spaces */
123     /* if old & new at same tab stop, its just difference */
124     if (tabs == 0) spaces = newpos - oldpos;
125     /* if not, then its remainder from nearest tab stop */
126     else spaces = newpos % TABSIZE;

128     for (i=0;i<tabs;i++) putchar('\t');
129     for (i=0;i<spaces;i++) putchar(' ');
130 }

132 /* true if wordstar control character */
133 boolean
134 iswscntrl(c)
135 int c;
136 {
137     if ((c == BOLD) ||
138        (c == UNDERSCORE) ||
139        (c == SUPERSCRIPT) ||
140        (c == SUBSCRIPT)) return(TRUE);
141     else return(FALSE);
142 }

144 newline()
145 {
146     if (!dotcmd) putchar('\n');
147     column = 0;
148     spacerun = 0;
149     dotcmd = FALSE;
150 }

152 wscntrl(c)
153 char c;
154 {
155     switch (c) {    /* print control character */
156     case BOLD:
157         if (bold) printf(BOLD_END);
158         else printf(BOLD_BEGIN);
159         bold = !bold;
160         break;
161     case UNDERSCORE:
162         if (underscore) printf(UNDERSCORE_END);
163         else printf(UNDERSCORE_BEGIN);
164         underscore = !underscore;
165         break;
166     case SUBSCRIPT:
167         if (subscript) printf(SUBSCRIPT_END);
168         else printf(SUBSCRIPT_BEGIN);
169         subscript = !subscript;
170         break;
171     case SUPERSCRIPT:
172         if (superscript) printf(SUPERSCRIPT_END);
173         else printf(SUPERSCRIPT_BEGIN);
174         superscript = !superscript;
175         break;
176     default:
177         /* unknown - ignore for now */
178         break;
179     }
180 }
```

COMPETITIVE EDGE

P.O. BOX 556 — PLYMOUTH, MI 48170 — 313-451-0665

CompuPro®, LOMAS, EARTH COMPUTER, MACROTECH, TELETEK

S-100 CIRCUIT BOARDS

CompuPro 286 CPU™	$750	Lomas Lightning 286™	$821	MACROTECH 286/Z80H	$795	TELETEK SYSMASTER II	$899
CompuPro MDRIVE-H®	$495	Lomas THUNDER 186™	$1195	EARTH TURBO SLAVE I 8HMz	$280	ILLUMINATED TECH. 1024x1024 Graphics	$995

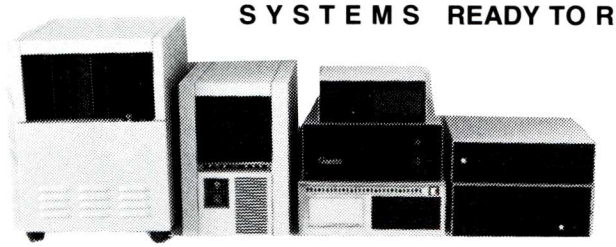
FREE MICRO/SYSTEMS JOURNAL SUBSCRIPTION WITH PURCHASE OF ANY OF ABOVE

DISK CONTROLLERS		CPU BOARDS		MEMORY BOARDS		I/O SERIAL/PARALLEL	
CompuPro Disk 1A™ A&T	$347	CompuPro 286™	$750	CompuPro RAM 22™ 120ns	$ 850	CompuPro System Support 1™	$245
CompuPro Disk 3™ A&T	$417	CompuPro SPU-Z™	$261	CompuPro Ram 23™ 64K 120ns	$ 277	CompuPro Interfacer 4™	$245
LOMAS LDP 72 Floppy Contr	$220	CompuPro 8085/88™	$245	CompuPro RAM 23 128K 120	$ 487	CompuPro Interfacer 3™	$346
Lomas WD1002 HD Contr	$325	CompuPro CPU Z™	$179	Lomas RAM 67 128K 100ns	$ 725	Lomas 2 ser, 2 par, clock Hazitall	$275
Teletek FDC II Flpy cont	$245	Lomas Lightning 286	$821	Lomas Megaram 256K 150 ns	$ 556	Lomas Octaport 8 serial	$316
Teletek HD H.D. Contr.	$375	Lomas 8086 8MHz	$420	Lomas Megaram 512K 150ns	$ 821	Lomas Octaport 4 serial	$200
80287 FOR LOMAS 286	$395	Lomas 8086 10MHz	$520	Lomas Megaram 1024K 150ns	$1096	Teletek PS10 2 par, 4 ser	$215
80287 FOR COMPUPRO 286	$325	Macrotech 286/Z80	$795	Lomas Megaram 2048K 150ns	$1595	Teletek I square 1 port	$195

SINGLE BOARD COMPUTERS		HARD DISKS		FLOPPY DRIVES		S-100 COLOR GRAPHICS	
THUNDER 186 256K QTY 1	$1195	Maxtor 2085 30 ms	$2295	8" Mitsubishi 2894-63	$389	Lomas Color Magic 16K	$476
THUNDER 186 256K Qty 3 up	$1095	Maxtor 1105 30 ms	$2795	5" Mitsubishi 4853 96 TPI for CompuPro	$150	Lomas Color Magic 32K	$556
TELETEK Systemaster®	$ 557	Rodime 53/40 55 ms	$1395	5" Mitsubishi 4851 48TPI	$150	CompuPro PC VIDEO 16K	$371
Teletek SBC-1 6MHz 128K	$ 695	Quantum Q540 45 ms	$1395	5" National JA551-2 48 TPI for Lomas	$135	Illuminated Tech. Color Graphics	$995
Teletek SBC-II Dual Slave	$ 854	Seagate 25/20 80 ms	$ 695	5" TEAC 55B for IBM PC®	$135	I.T. Board is 1024 x 1024 and	
TELETEK SYSTEMASTER II®	$ 899	Fujitsu 13/10 90 ms	$ 395	5" Dbl. Sided Color Diskettes 10 Pk	$ 21	runs Auto-CAD-2 & other on Lomas	
EARTH TURBO SLAVE I 8MHZ Z80	$ 399	Maxtor 1140 30 ms	3595	5" Dbl. Sided Color Diskettes 100 Pk	$199		

Illuminated Technology 1024 x 1024 Color Graphics Board (for AutoCAD2)™ $995

SYSTEMS READY TO RUN



NV MEMORY DRIVE from LOMAS

Direct Replacement for CompuPro!
Battery Backup OPTION!

512 K $521 2048K $1496

CompuPro 85/88, 256K, CDOS, SS1, I/O 4, 2-96TPI DRS, 10 Slot $3095
CompuPro 85/88, 256K, CDOS, SS1, I/O 4, 1-96TPI, 20MB, 10 Slot $4295
286/Z80H, 1024K Static, CDOS, SS1, I/O 4, 1-96TPI, 40MB, 10 Slot $7495
MDRIVE-H option $495 per 512K or $1795 per 2048K each
Lomas 286, 1024K, 20MB HD, 1-5", CDOS, 6 SERIAL, 2 PAR, 10 Slot $4995
Lomas Thunder 186, 256K, 20MB HD, 1-5", CDOS, 4 Slot $2895
Teletek 8MHz Master, 4-8MHz 128K SLVS, 1-5", 20 MB HD, TDOS $4495

LOMAS 24 MEGA BYTE TAPE BACKUP Sub System for MOST CP/M-86 & MSDOS S-100 Systems $1495

Independent, Custom, Scientific Programming Service Available

UPGRADE YOUR IBM PC!!

CABINETS		MONITORS		GRAPHIC BOARDS	
CompuPro Enclosure 2™ Desk	$715	Amdek 310A	$159	Hercules Monochrome	$299
CompuPro Enclosure 2 Rack	$750	Taxan Color 440	$549	Hercules Color Card	$159
Para Dynamics CVT DR Cab 2-8" Desk	$395	Princeton Color HR-12	$459	Tecmar Graphics Master	$449
Para Dyn CVT Rack DR Cab 2-8"	$495	Princeton Color SR-12	$649	Paradise Graphics	$279
Para Dyn CVT Desk DR Cab 1-8", 5" HD	$425	**MULTI-FUNCTION BOARDS**		STB Graphix + II	$279
Para Dyn Roll A Round CVT Pronto	$1195	AST 6 Pak 64K	$245	**FLOPPY DRIVES**	
Para Dyn 10 Slot Mini Pronto CVT	$795	Quadram Expanded Quadboard OK	$219	TEAC 1/2 HT FD55B	$119
Single 5" H.D. Cabinet	$150	TECMAR Captain 64K	$199	Mitsubishi 96 TPI	$125
Two 5" Hard Disk Cabinet	$195			5" DSDD Color Diskettes	$21

ALL PRICES & SPECIFICATIONS SUBJECT TO CHANGE AND STOCK ON HAND

Turbo Pascal Corner

by David W. Carroll

Borland International revolutionized the microcomputer world in October 1983 by releasing the **Turbo Pascal** compiler for $49.95. Since that time, the Pascal language has undergone a rebirth of interest — mostly due to the ease of writing and compiling programs with Turbo Pascal and the powerful extensions to the language it provides. Borland has sold over 250,000 copies of Turbo Pascal in about 18 months. In March, Borland announced Turbo Pascal version 3.0 with several extensions for $69.95.

This column will feature tips and techniques for using Turbo Pascal productively on MS-DOS and CP/M microcomputer systems. We will discuss typical problems and their solutions. Reader suggestions, comments, and questions are encouraged. You may address comments to:

Turbo Pascal Corner
P.O. Box 699
Pine Grove, California 95665

WordStar Conversion Utility

WordStar is one of the three programs to become a million seller, mostly due to its availability in the early days of microcomputing. Though many use WordStar, few understand how it actually works with text files on a character level. Our featured program this issue is a utility written in Turbo Pascal to convert WordStar files to standard ASCII files.

WordStar uses the "eighth bit" or parity bit of some characters in its text files to mark them for later format processing. The "standard" ASCII character set does not use the eighth bit. WordStar files may also include "dot commands" (lines beginning with a period) for format instructions and "print enhancement mode commands" (control characters like ^S for underline) mixed in with the text.

Normally, this is not a problem, but if you wish to transfer text files to other systems (word processors, typesetters, spreadsheets, databases) or send files via electronic mail (i.e. MCI Mail), you must first translate WordStar format files to "standard ASCII" files. Some of this translation can be done by using the WordStar Global Find and Replace function, but this is a time consuming and error prone process. The "hidden" or "soft" characters using the eighth bit are not easily modified by WordStar. In addition, some print control characters (like ^S) cannot be searched for in WordStar.

A solution is to write a text translation program in Turbo Pascal. One possible program is presented at the end of this discussion. It is not overly elaborate due to space constraints, but more options, menus, color displays, and so on are easily added. Text translation falls under the general category of filters, and this generalized program design may have many more applications than just WordStar format translation.

The Program

The main program is quite short, as is typical in Pascal. It displays a title and calls the menu, open files, process, and exit procedures. The Turbo Pascal window built-in procedure allows easy set-up of partial display screens and is used throughout the program.

The program does not contain any unusual data structures, but several arrays are used. The print mode control characters are in constant array **spc** and their "plain text" string substitutes are held in two more constant arrays — **pcon** for beginning the print mode and **pcoff** ending the print mode. The variable array **flagpc** contains boolean values indicating whether a given print mode is currently active.

The variable array **numlist** is used to hold the byte values of the characters in each input line. Using an array containing the byte value of each character cuts down on the number of type conversions needed in the translation portion of the program.

The **menu** procedure uses a simple display format and a yes/no input function — **inyn** to return a boolean value for each translation option. This function could be included in the **menu** procedure, but is left as a global function for future growth.

Open_files is a fairly standard open file procedure with error checking. It checks for a pre-existing output file and requires user confirmation before overwriting the file.

Procedure **process** keeps track of lines and characters processed and calls procedures **get_line**, **test_line**, and **translate_line**.

The **get_line** procedure inputs each character's byte value into the line array. Each line is terminated by detecting a high bit masked line feed character. A simpler version of the program could process one character at a time. However, in this example, an entire line is processed at one time to allow advanced parsing functions to be built onto this basic program structure. Some types of translations require contextual look-ahead and look-back capability.

Input lines beginning with a "dot" or period can be completely ignored if the "Strip dot commands?" menu option is selected. This operation is handled by the **test_line** procedure, which sets variable **translt** to false if the line should be skipped.

The actual translation is done by the procedure **translate_line**. Tests are performed on the byte array values and the values are translated as required. Finally non-null characters are written to the output file. The only unusual operation is the substitution of print mode commands, where starting and ending modes must be detected using **flagpc** and the appropriate string substituted for the WordStar control character (i.e. ^S becomes either [us] or [eus]).

The easiest part of the translation is to strip the "high" or eighth bit off each character. This can be done by logical and-ing each character's value with 127 (hex 7f) as shown by this simplified fragment:

```
read(infile,ch);                {read character}
chnum := ord(ch);               {get its value}
lonum := chnum and 127;         {strip high bit by anding}
write(outfile,chr(lonum);       {write character}
```

Most other translation operations are performed by simple tests and substitutions.

The program can be speeded up by eliminating the screen display of lines and characters processed, or showing only the final counts. You are encouraged to modify it for your own requirements.

This program, its "big brother", and other Turbo Pascal public domain programs are available 24 hours a day for free downloading on the "High Sierra RBBS" system at 209/296-3534.

David W. Carroll is a freelance writer and computer consultant living in the Sierra Nevada foothills near Sacramento, California. He is the author of **Telecommunications with the IBM PCjr** published by Prentice Hall in March 1985 and **Programming with Turbo Pascal** to be published by McGraw Hill this summer.

```
program wsutil;

{WordStar file utility -- 3-27-85 Ver B5 }
{Copyright 1985 by David W. Carroll }
{ }
{Converts WordStar files to ASCII text files }
{with options. }

const
   nummax = 200; {Maximum input line length}

   version = 'B5';
   date = 'March 27, 1985';

   null = 00;
   bell = 07;
   lf = 10;
   ff = 12;
   cr = 13;
   nobrksp = 15; {no break space ^O}
   softhyph1 = 30; {mid-text soft hyphen}
   softhyph2 = 31; {eol soft hyphen}
   space = 32;
   hyph = 45;
   period = 46;
   softlf = 138; {soft line feed}
   softcr = 141; {soft carriage return}
   softsp = 160; {soft space}

   ctlb = 02;
   ctld = 04;
   ctls = 19;
   ctlt = 20;
   ctlv = 22;

 arrays pcon and pcoff contain the strings that are}
 substituted for WordStar print control characters}
 ^b, ^d, ^s, ^t, and ^v. Constant pc is the total }
 number of WS print control characters supported. }

   pc = 05;
   spc: array[1..pc] of byte = (ctlb, ctld, ctls, ctlt, ctlv);
   pcon: array[1..pc] of string[10] =
        ('[bf]','[ds]','[us]','[sup]','[sub]');
   pcoff: array[1..pc] of string[10] =
        ('[ebf]','[eds]','[eus]','[esup]','[esub]');

var
   infile : text;
   outfile : text;
   numlist : array[1..nummax] of byte;
   flagpc : array[1..pc] of boolean;
   totchrin : real;
   totchrout : real;
   cnt : integer;
   quit : boolean;
   stripsp : boolean;
   striplf : boolean;
   chgpc : boolean;
   stripdc : boolean;
   strippc : boolean;
   translt : boolean;

procedure open_files;
var
   infname : string[20];
   outfname : string[20];
   ans : string[10];
   goodfile : boolean;

begin
     window (1,5,80,25);
     repeat
        ClrScr;
        write ('Input filename --> ');
        readln (infname);
        assign(infile,infname);
        {$I-} reset(infile) {$I+};
        goodfile := (IOresult = 0);
        if not goodfile then
        begin
          write (chr(bell));
          writeln ('FILE ',infname,' NOT FOUND');
          delay(3000)
        end;
     until goodfile;
     window (1,6,80,25);
     repeat
        ClrScr;
        write ('Output filename --> ');
        readln (outfname);
        assign (outfile,outfname);
        {$I-} reset(outfile) {$I+};
        goodfile := (IOresult <> 0);
        if not goodfile then
        begin
          write (chr(bell));
          write ('FILE ',outfname,' EXISTS,OVERWRITE? (y/n)');
          readln (ans);
          goodfile := (UpCase(ans[1])='Y')
        end;
     until goodfile;
     rewrite(outfile)
end;

procedure get_line;
var
   ch : char;
   num : byte;
   lonum : byte;

begin
  ch:=chr(0);
  lonum:=0;
  num:=0;
  cnt:=0;

  while not eof(infile) and (lonum<>lf) do
  begin
    cnt:=cnt+1;
    read(infile,ch);
    totchrin := totchrin + 1;
    num:=ord(ch);
    lonum:=(num and 127);
    numlist[cnt]:=num;
  end
end;

procedure test_line;
begin
     translt := true;
     if stripdc then
       if numlist[1]=period then translt := false;
end;

procedure translate_line;
var
   spstr : string[10];
   indx1 : integer;
   indx2 : integer;
   indx3 : integer;
   num : byte;
   chnum : byte;
   lonum : byte;
   exch : boolean;

begin
  for indx1:=1 to cnt do
  begin
    exch := false;
    num:=numlist[indx1];
    chnum := num and 127;
    lonum :=chnum;

    if (num=softhyph2) then
       chnum := hyph
    else if (num=softhyph1) then
       chnum := null;

    if num=nobrksp then chnum := space;

    if chgpc then
    begin
      for indx2:=1 to pc do
      begin
      if lonum = spc[indx2] then
      begin
        chnum := null;
        exch := true;
        if flagpc[indx2] then
          spstr := pcoff[indx2]
        else
          spstr := pcon[indx2];
        flagpc[indx2] := not flagpc[indx2]
      end
    end
  end;
```

```
    if stripsp and (num=softsp) then chnum := null;
    if striplf and (lonum=lf) then chnum := null;

    if strippc then
      for indx3 := 1 to pc do
        if lonum = spc[indx3] then chnum := null;

    if chnum <> null then
    begin
      write (outfile, chr(chnum));
      totchrout := totchrout+1
    end;
    if exch then
    begin
      write(outfile,spstr);
      totchrout := totchrout + length(spstr)
    end
  end
end;

function inyn : boolean;
var
  ans  :  string[10];

begin
  write('[y/n]  ');
  readln(ans);
  inyn := (Upcase(ans[1]) = 'Y')
end;

procedure menu;
begin
      writeln;
      writeln('Wordstar to ASCII Conversion');
      writeln;
      writeln;
      write(' 1.   Strip soft-spaces (un-justify)? ');
      stripsp := inyn;
      write('  2.      Strip  line  feeds?    ');
      striplf := inyn;
      write(' 3.  Change control (print) commands? ');
      chgpc := inyn;
      write('  4.      Strip  dot  commands? ');
      stripdc := inyn;
      if chgpc = false then
      begin
        write('  5.     Strip  print  commands? ');
        strippc := inyn;
      end
      else
        strippc := false;
      writeln;
      write('         Quit?            ');
      quit := inyn;
end;

procedure process;
var
   line  :  integer;
   indx  :  integer;

begin
  window(1,7,80,25);
  ClrScr;
  line:=0;
  totchrin:=0;
  totchrout:=0;

  for indx :=1 to pc do
    flagpc[indx] := false;

  while not eof(infile) do
  begin
    line:=line+1;
    get_line;
     test_line;
     if translt then
      begin
       translate_line;
       window(1,12,80,16);
       ClrScr;
       writeln('Line  #  ',line:5);
       writeln('Total characters input: ',totchrin:6:0);
       writeln('Total characters output: ',totchrout:6:0);
       writeln('Total filtered(+)/added(-):
                                    ',(totchrin-totchrout):
      end
    end
end;
procedure exit;
begin
     window(1,23,80,25);
     ClrScr;
     writeln('Translation  completed!');
     writeln(outfile);
     close(infile);

     close(outfile)
end;

begin
     ClrScr;
     writeln;
     writeln('WordStar File Conversion Program');
     writeln('Copyright 1985 by David W. Carroll');
     writeln('Version  #',version,' of  ',date,'.');
     writeln;
     window(1,5,80,25);
     ClrScr;
     menu;
     ClrScr;
     if not quit then
     begin
       open_files;
       process;
       exit
     end
     else
       writeln('Translation  cancelled.');
end.
```

TAKE THE PAIN OUT OF BACKUP

Famous Excuses For Not Doing Backups

Backing up a Winchester onto multiple floppies takes forever.

PIP makes you remember what you changed — too much trouble.

Disk failures only happen to the other guy.

Floppy-To-Floppy Incremental Backup For Only $40.

Like PIP but copies just the files that have changed.

Order Qbax1.

The Remedy: Copy Just What's New — Automatically!

Qbax2 will:
Split files bigger than one floppy.
Track all the pieces.
Tell you which floppy it needs and when.
Give you a built-in catalogue.
Reclaim wasted floppy space.

Qbax2 only $95 from:

Amanuensis, Inc.
R. D. 1, Box 236
Grindstone, PA 15442
(412) 785-2806

For CP/M 2.2 on 8" SSSD and popular 5¼" formats. Shipping: $2 U.S. and Canada, $4 overseas. Qbax TM Amanuensis, Inc. CP/M® Digital Research.

FOR EVERYONE WITHOUT A DISPENSATION FROM MURPHY'S LAW

THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

SERVICES

- **Programmer's Referral List**
- **Compare Products**
- **Help find a Publisher**
- **Evaluation Literature free**
- **BULLETIN BOARD - 7 PM to 7 AM 617-826-4086**
- **Dealer's Inquire**
- **Newsletter**
- **Rush Order**
- **Over 700 products**

Free Literature - Compare Products

Evaluate products Compare competitors Learn about new alternatives. One *free call* brings information on just about any programming need. Ask for any ***"Packet"*** or "Addon Packet". ☐ ADA, Modula ☐ "AI" ☐ BASIC ☐ "C" ☐ COBOL ☐ Editors ☐ FORTH ☐ FORTRAN ☐ PASCAL ☐ UNIX/PC or ☐ Debuggers, Linkers, etc.

RECENT DISCOVERIES

SMALL TALK for PCDOS - "Methods" has objects, windows, browser, inspector. PCDOS $239

"C" LANGUAGE

	OUR PRICE
MSDOS: C86-8087, reliable	call
Instant C - Inter., fast, full	495
Lattice C - the standard	call
Microsoft C 3.0 - new	279
Williams, debugger, fast	call
CPM80: EcoPlus C - faster, SLR	275
BDS C - solid value	125
MACINTOSH: Hippo II	375
Megamax - optimizer, full	275
Consulair's MAC C	275

Compare, evaluate, consider other Cs

BASIC

	RUNS ON	
Active Trace-debug	86/80	75
BASCOM-86 - MicroSoft	8086	279
BASIC Dev't System	PCDOS	115
BetterBASIC - 640K	PCDOS	185
CB-86 - DRI	CPM86	419
Prof. BASIC Compiler	PCDOS	89
CADSAM-Full Btree, source	MSDOS	150
SCREEN SCULPTOR	PCDOS	115

Ask about ISAM, other addons for BASIC

SERVICE

ALL PRODUCTS - We carry 700 products for MSDOS, CP/M 86, CP/M 80, Mac-Intosh and key products for other micros.

EDITORS Programming

	RUNS ON	OUR PRICE
BRIEF - Intuitive, flexible	PCDOS	195
C Screen with source	86/80	75
Epsilon - like EMACS	PCDOS	195
FINAL WORD-for manuals	86/80	215
PMATE-powerful	8086	185
VEDIT-full, liked	86/80	119
XTC - multitasking	PCDOS	95

COBOL

Dig. Res-decent	MSDOS	500
Macintosh COBOL-Full	MAC	1850
MBP-Lev. II, native, screen	MSDOS	885
Micro Focus Prof.-Full	PCDOS	call
Microsoft-Lev II, no royal	MSDOS	500
Ryan McFarland-portable	MSDOS	695

LANGUAGE LIBRARIES

GRAPHICS: GraphiC-source in C	MSDOS	219
GRAPHMATIC-3D, FTN, PAS	PCDOS	125
HALO-fast, full-all lang.	PCDOS	139
FILE MGNT: BTrieve-all lang.	MSDOS	215
CIndex + -source, no royal.	86/80	369
CTree-source, no royal.	ALL	369
dBC ISAM by Lattice	8086	229
dB VISTA-"Network" Structure	MSDOS	465
PHACT-up under UNIX, addons	MSDOS	225
OTHER: CUtil by Essential	MSDOS	129
Greenleaf - 200 +	MSDOS	159
CSharp - **Real-Time**	MSDOS	600
PORTABLE C to PC, Mac, II	Many	125
SOFT Horizons - Blocks I	PCDOS	139
SCREEN: CURSES by Lattice	PCDOS	125
CView - input, validate	PCDOS	195
MetaWINDOW - icons, clip	PCDOS	139
PANEL - many lagg. term	MSDOS	249
ProScreen - windows, source	PCDOS	415
Windows for C	MSDOS	175

FORTRAN

	RUNS ON	OUR PRICE
MS FORTRAN-86 - Impr.	MSDOS	$ 239
DR Fortran-86 - full '77'	8086	249
PolyFORTRAN-XREF, Xtract	PCDOS	165

OTHER PRODUCTS

Assembler & Tools - DRI	8086	159
Atron Debugger for Lattice	PCDOS	395
cEnglish - dBase to C	MSDOS	750
C Helper: DIFF, xref, more	86/80	135
CODESMITH-86 - debug	PCDOS	139
MacASM-full, fast, tools	MAC	115
MBP Cobol-86 - fast	8086	885
Modula 2 for	MAC, PCDOS	90
Micro: SubMATH-FORTRAN full	86/80	250
Microsoft MASM-86	MSDOS	125
MSD Debugger	PCDOS	119
Multilink - Multitasking	PCDOS	265
PC FORTH + -well liked	MSDOS	219
PFIX-86 Debugger	MSDOS	169
PL/1-86	8086	495
Polylibrarian - thorough	MSDOS	95
PolyMAKE	PCDOS	95
PROFILER by DWB - flexible	MSDOS	109
Prolog-86-Learn, Experiment	MSDOS	125
SYMD debugger-symbols	PCDOS	119
TRACE86 debugger ASM	MSDOS	115

Call for a catalog, literature, and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™

128-M Rockland Street, Hanover, MA 02339

Visa Mass: 800-442-8070 or 617-826-7531 MasterCard 8517

Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and **POs**. All formats available.

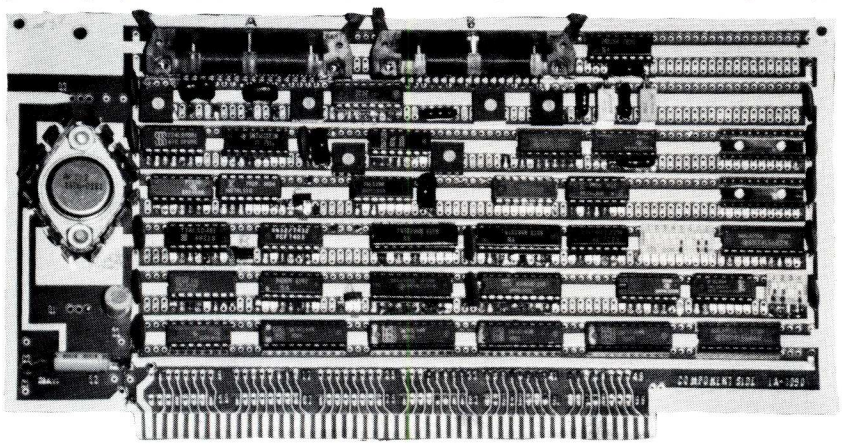
Build an S-100 to PC Bus Converter

by John Monahan

Build an interface so you can use PC video, disk controller and I/O boards with your S-100 system.

Having been an S-100 computer user for the past 8 years I have always taken joy in the fact that I had a better computer system than many "ready made" computers that appeared and later disappeared over the years. As more memory, floppy disk drives, and hard disks began to appear as essential components for a good system I simply shopped around for the appropiate S-100 boards. During that time I had written my own personalized monitor to do all those things one wants a computer to do exactly as I wanted them done. Over the years in fact I have built up a complicated web of hardware units that involve: a second S-100 slave computer to handle multi-printer I/O, speech synthesis, keyboard single key to multikey translations, numerous video boards as well as a master/slave arrangement for Z80 and 8086/8087/8089 CPU control transfer. This rather complicated system is driven by CPM80+, CPM86, and MSDOS disk operating systems. There is still no doubt in my mind that the S-100 bus is the best suited for my needs. Because it is a 16-bit bus it is faster than the IBM-PC.

As time went by a serious limitation of the S-100 bus became apparent. Computer board manufacturers were no longer supporting the bus as much as they did in the past. Perhaps the best example of this was the long delay in bringing out an S-100 video board that was both software and hardware compatable with the PC. The cruel hard fact of life is that the PC market is larger than the S-100 market, so many board manufacturers rushed into that arena. This has led to a good supply of many types of PC bus boards (memory, I/O, disk controllers, video, clock boards to name only a few). A second result of this interest in the PC bus is that the almost oversupply of manufacturers has lead to a supply of good boards at very low prices.



The S-100 to PC Bus Interface board.

Why build a converter?

As I watched this drift away from the S-100 bus, I wondered how many companies would be smart enough to avoid the oversupplied PC market. I waited for them to expand their S-100 bus board set. In particular I was looking for an PC compatible video board. I waited and waited and waited.... Nothing happened. In desperation I considered putting together my own S-100 video board. A careful examination of IBM's technical manuals revealed that this would not be all that simple. All those clock circuits scared me off. Besides I would have to really build two boards. One for monochrome and one for graphics.

Then I had an idea. If I could not get an PC compatible video board for the S-100 bus, why not make a board that would convert S-100 bus signals into a form that could be utilized by PC's boards. In other words make an S-100 to PC-bus converter board.

The crude outline for the project started to take shape in my mind. I would build one S-100 board that would connect the S-100 bus over a buffered ribbon cable to a second 62-

pin bus (on its own motherboard) that is compatible with the IBM-PC bus. In this way my S-100 system would talk to PC cards as if they are on the S-100 bus itself. In fact, from a software point of view the 8086 would not know on which hardware bus the board resides.

As new PC boards arrive, I simply plug them into my extended bus. I now have the best of both worlds. My old S-100 system with most of its memory in fast static RAM (16 bit), its hard disk and memory disk etc. all well oiled and at the same time it has the capability to take on PC boards. While I have many boards in my system that I have constructed and would like to talk about, it is the S-100/PC converter board I would like to describe here.

Building the converter

To begin with, this is not a construction project for a novice. There are bus drivers on both bus interfaces of this board. Incorrect activation of these chips could severly damage your system. Only take on this project if you are sure you understand what you are doing. Perhaps this article may interest a hardware manufacturer to contact me to mass produce the board in which case you will not have to put in the hours of work required to carry off such a project.

Finally, dynamic memory on some PC memory boards is not refreshed by a controller on the memory board itself but rather by the cycling of a 8237 DMA controller on the system board. This continuous reading of RAM once every 72 clock cycles (7% of bus bandwidth) keeps the dynamic RAM refreshed. At present I do not have a DMA controller on my PC motherboard to refresh RAM since I have no need for one. Should you wish to interface (slow) PC compatible memory boards with your S-100 system you will have to address this problem. The main use of this converter board is to interface PC video boards, disk controllers, I/O boards and the like.

The PC bus signals

To understand how this board works we will have to examine all the PC signals and describe how this board simulates them via their S-100 counterparts. The S-100 signals are well described in Sol Libes & Mark Garetz's book "Interfacing to S-100/IEEE 696 Microcomputers". The PC signals are well described in the IBM-PC Technical Reference Manual (In my 1981 edition, pages 2-9 to 2-12). The PC bus consists of 62 pin connectors arranged as 2 rows of 31 pins each. One side is numbered A1 to A31. The other is B1 to B31. These are:

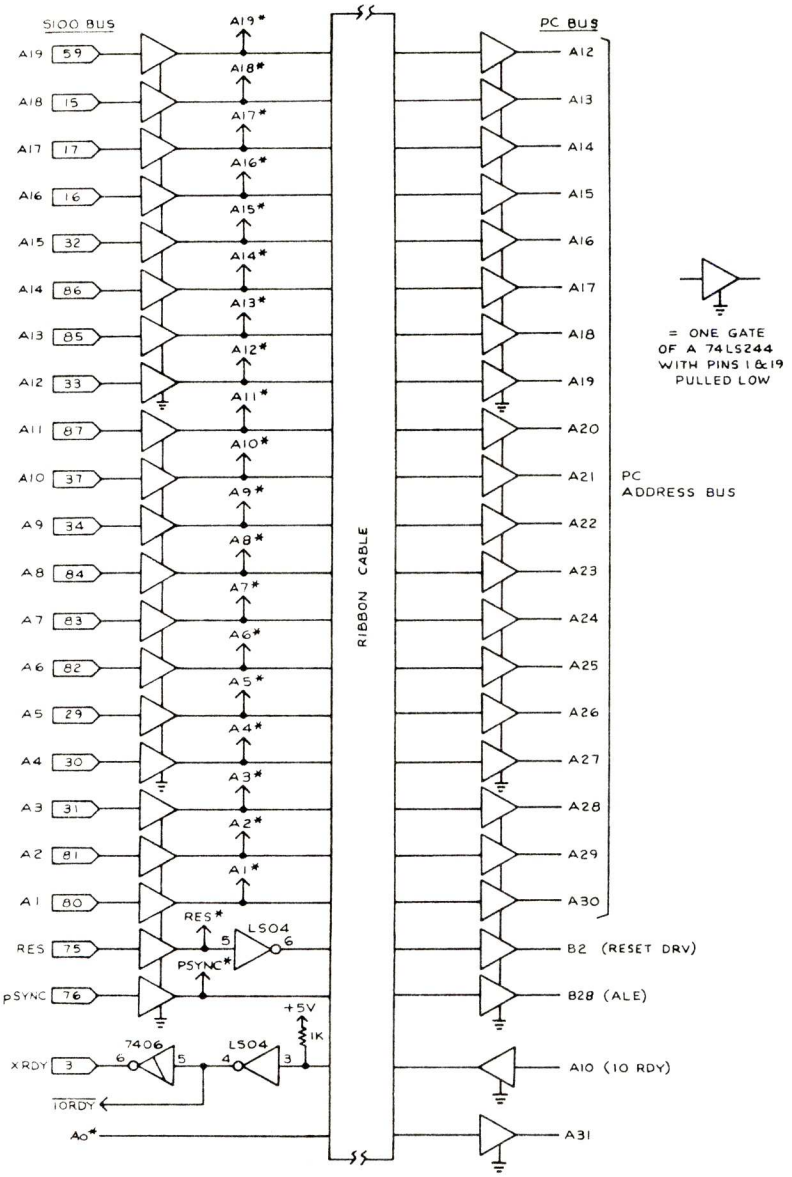


FIGURE 1

A1 -I/O CH CK Provides the CPU with parity error information if an error is detected on any IBM card. It is normally high and is only pulled low when an error is detected. This signal is simply passed directly on to S-100 bus via NMI int line (pin 12) using an open collector output (figure 4).

A2 - A9 Data bits 0 to 7 These lines pass 8 bit data to and from PC cards. Because PC uses an 8088, all data is 8 bits wide. 16 bit transfers are sent or received as two 8 bit data units by the processor. Our first complication is that I use an 8086 on my S-100 bus. For reasons I do not want to go into here, I do not have the capability on the 8086 board of talking to 8 bit memories. It will be clearly necessary on our converter board to convert 16 bit data to 8 bit units to interface to the PC data lines. I also have an old (backup) S-100 8088 board, the converter will also work with this board in 8 bit mode (ie. the S-100 sXTRQ* is not used).

A10 +I/O CH RDY This line (normally high or "ready") is pulled low by a memory or I/O device to lengthen CPU access time to the device. In particular it is used by PC dynamic memory boards to refresh their memory cells. It turns out that this is perhaps the most critical line to be interfaced to the S-100 bus. I will discuss it in more detail later. Suffice to say that the S-100 bus must never access or leave the PC bus when this line is low (i.e. not ready).

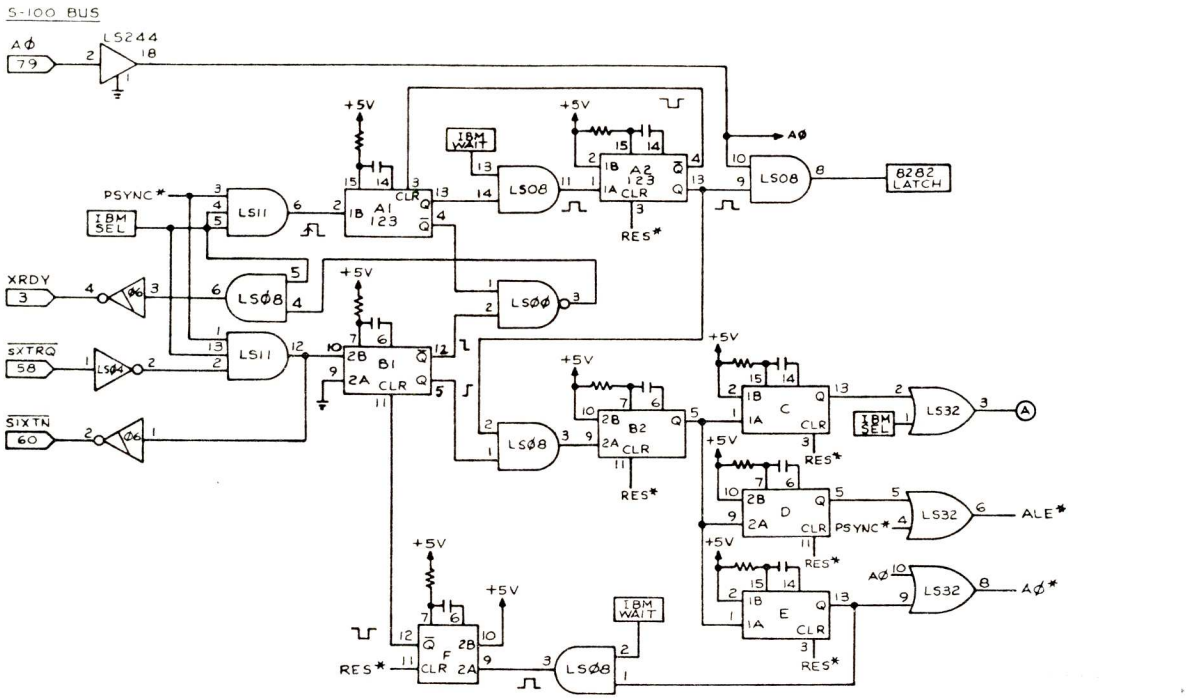


FIGURE 2

A11 +AEN Indicate when a DMA controller has control of bus. Because I do not implement DMA that originates on the PC bus, line is not used and is tied low.

A12-A31 A0 - A19 The address lines on bus to address memory and I/O devices. They are directly comparable with the S-100 address lines and are connected directly to both buses via drivers.

B1 This is one of the bus ground lines and is connected directly to the S-100 ground lines (pins 20,50,53,100).

B2 RESET DRV Used to reset or initialize system logic on the bus. It is active high. The S-100 RESET line (pin 75) is active low, so we must invert the S-100 signal on the converter board before connecting it to the PC bus.

B3 +5V Unlike the S-100 bus the PC bus provides filtered 5 volts directly to its cards. Because a number of cards may be on the bus, the supply current may be considerable. I supply 5 volts to the PC bus from a 5 Amp +5 voltage regulator placed on the PC motherboard. This is enough for a few cards but may need to be increased later.

B4 IRQ2 This is one of 6 Interrupt request lines used to signal the CPU that an I/O device needs some sort of attention. An interrupt is generated by raising IRQ2 line and holding it high. The S-100 bus has interrupt request lines also. However they are active low. These signals are jumperable on converter card and inverted.

B5 -5VDC A -5 volts supply provided to PC cards that require this voltage. A simple 1 amp voltage regulator on PC motherboard is sufficient.

B6 +DRQ2 The DRQ lines on PC bus are used by devices such as a DMA controller to request control of bus away from 8088. The S-100 bus also has DMA request lines. However a DMA request originating on PC side of converter bus could become a can of worms. For the moment I choose to ignore these lines.

B7 -12VDC A -12 volt supply provided to PC cards that require this voltage. A simple 1 amp voltage regulator on the PC motherboard is sufficient.

B8 We get off light for this one

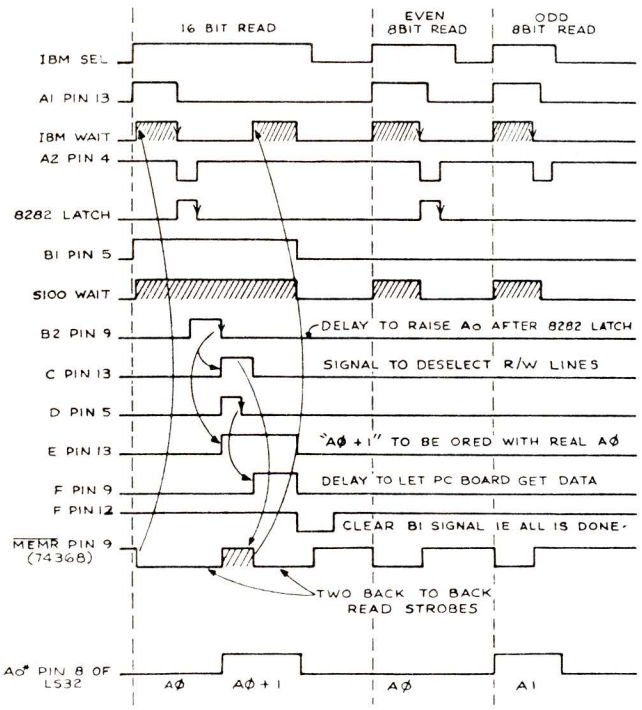


FIGURE 6

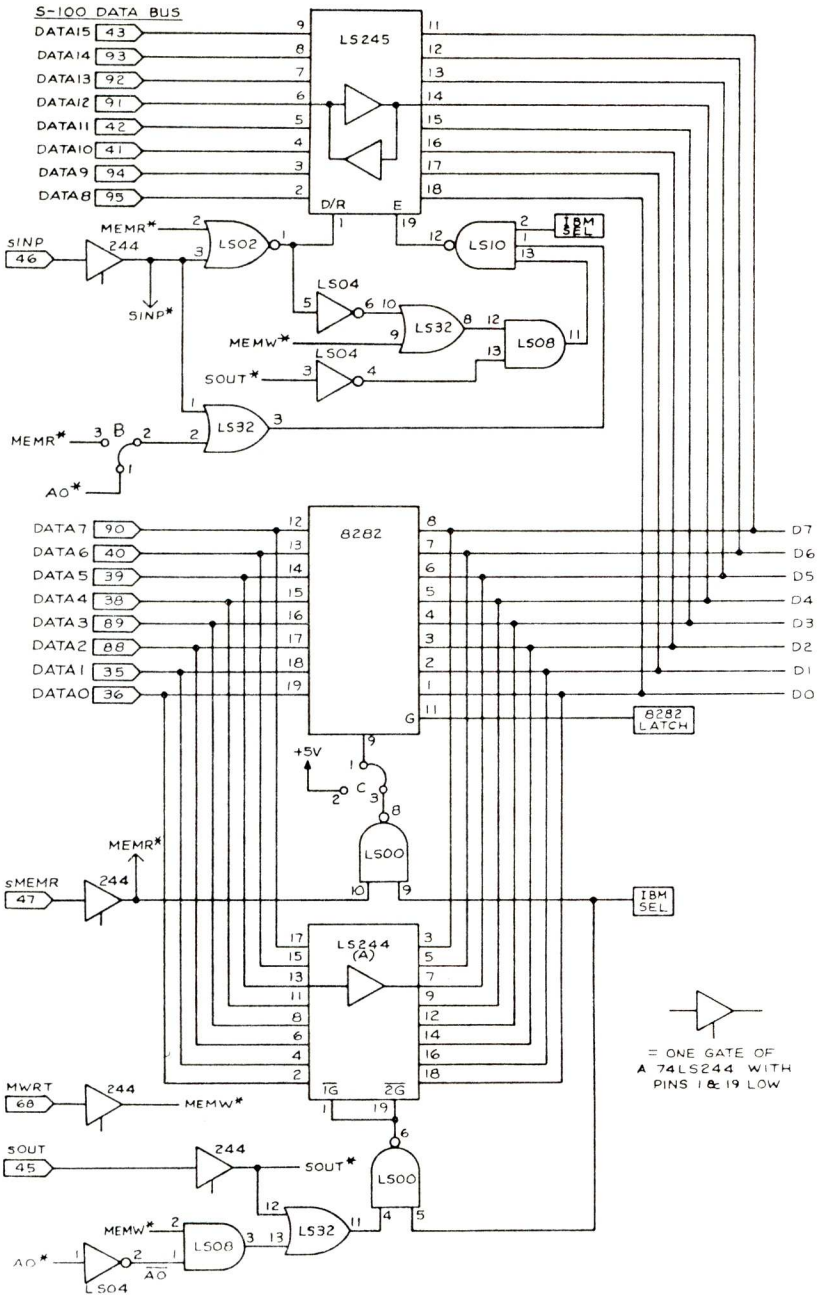


FIGURE 3

since no function has yet been assigned to this pin by IBM.

B9 +12VDC A +12 volt supply provided to PC cards that require this voltage. A simple 1 amp voltage regulator on the PC motherboard is sufficient.

B10 GND Same as B1.

B11 -MEMW Instructs memory devices to store data that is on data bus (active low). Corresponding S-100 signal is MEMW (pin 68). Since it is active high it must be inverted.

B12 -MEMR Instructs memory devices to read data that is on data bus (active low). The corresponding S-100 signal is sMEMR (pin 47). Since it is active high it must be inverted.

B13 -IOW Instructs I/O devices to store data that is on data bus (active low). Corresponding S-100 signal is sOUT (pin 45). Since it is active high it must be inverted.

B14 -IOR Instructs I/O devices to read data that is on data bus (active low). Corresponding S-100 signal is sINP (pin 46). Since it is active high it must be inverted.

B15 -DACK3 One of the 3 bus DMA acknowlege lines. Goes low when 8088 grants a DMA request. The S-100 bus has DMA acknowledge lines also. However I have not utilized them.

B16 +DRQ3 Same as B6.

B17 -DACK1 Same as B15.

B18 -DRQ1 Same as B6.

B19 -DACK0 Same as B15. Used by PC to refresh system dynamic memory.

B20 CLK This is the IBM-PC bus system clock (4.77 MHz , 33% duty cycle). Since the system clock on my S-100 system will normally be running at a very different frequency (8 MHz at present), it is clearly necessary to generate a separate system clock on the PC motherboard. The simplest way to do this is copy the PC directly by using an 8284A clock generator (figure 4), insuring that timing on the bus will be exactly as in a PC.

B21-B25 IRQ7-IRQ3 More interrupt lines; see B4.

B26 -DACK2 Same as B15.

B27 +T/C Provides a pulse when terminal count for any DMA controller is reached; not used in my system.

A28 +AEN The address latch enable signal produced by the PC 8288 Bus Controller chip. The falling edge of this signal indicates that a valid address is on the bus. The S-100 pSYNC (pin76) signal is quite compatible with this signal.

A29 +5VDC Same as line B3.

A30 +OSC A high speed clock with a 70 nsec. period (14.31818 MHz) and a 50% duty cycle. Since this signal is used for many video boards it must be accurate. It is generated exactly as IBM does using an 8284A clock generator directly on the PC motherboard.

A31 +GND Another ground same as B1.

The above describes all PC bus signals. Now let us see how we connect them to the S-100 bus.

Figure 1 shows the connections for the S-100 address lines. Because all signals are being sent over a ribbon cable (in my case 2 feet long), it is necessary to buffer signals at both ends of the cable. I use 74LS244 line drivers with pins 1 and 19 tied to ground to enable the devices. Also, just to be on the safe side I run a ground line between each address line in the cable. The address lines with stars above them (eg. A19*) are simply S-100 signals buffered through the 74LS244. These are used elsewhere on the board. This insures that each S-100 bus signal has only a single gate load. It is very confusing at first glance to see the PC address pins A12 to A30 refer to the pin numbering system not to the value of the address line itself.S-100 pin 75, RESET is inverted through a 74LS04 before becoming RESET DRV on the IBM bus. pSYNC becomes ALE. IO RDY on the IBM bus is inverted and

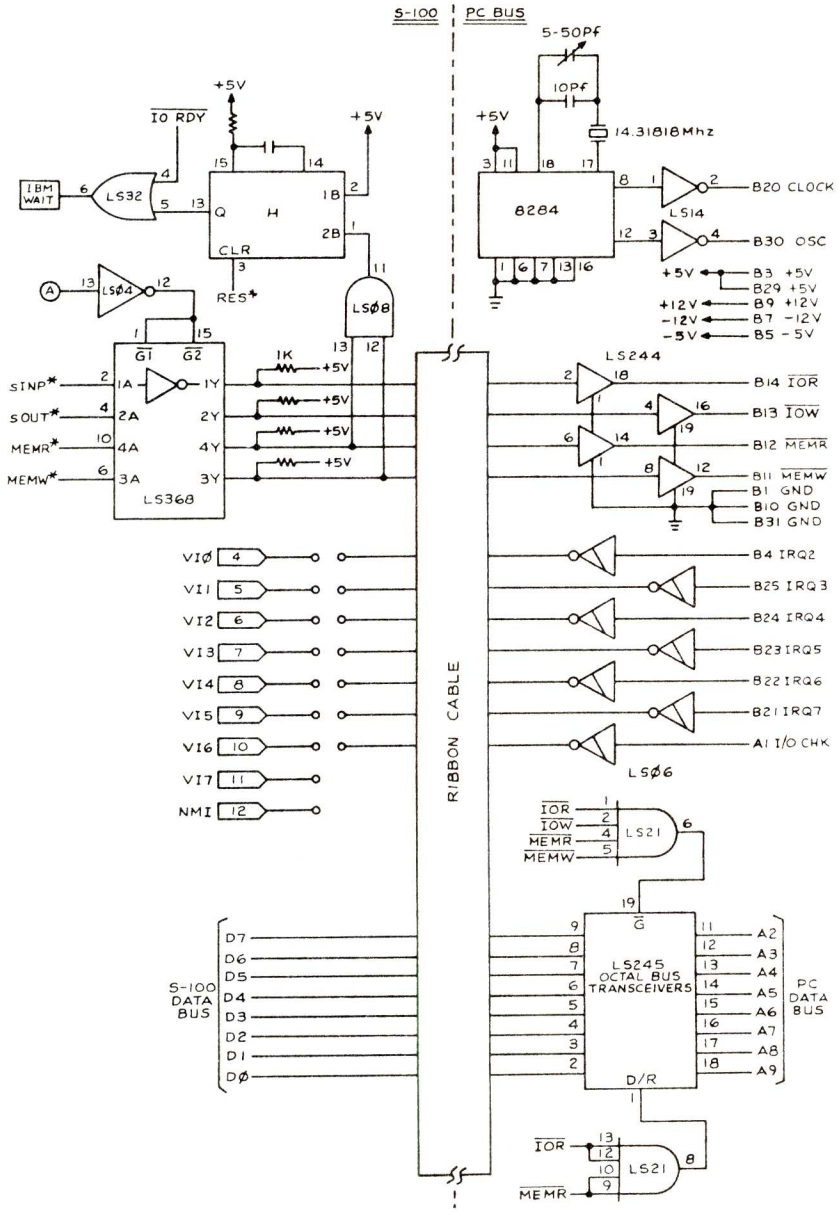


FIGURE 4

then passed on to an open collector inverter 7406 to become XRDY. Address line A0 from the S-100 bus is processed in a special way (see below) to become A0*. This is sent to address line 0 on the PC bus (pin A31).

Figure 2 is the heart of the circuit. Let us consider first 8 bit memory read or write. If the IBM select circuit (described below) determines that the S-100 CPU is addressing a board in one of its PC windows, the Q1 output of one-shot A1 will go high after pSYNC goes high. At the same time the not Q output of A1 will go low holding the S-100 bus in a permanent wait state. The read/write strobe going to the PC board (figure 4) will cause IBM WAIT to go high. When this is over (and it is important to remember that this is highly variable, depending on for example, status of 6845 on IBM video board), the input to A2 will go low. At this point there is valid data on the IBM bus. A2 output Q latches the data for an 8282 (described below) and A2 not-Q output clears single shot A1, which removes the wait state from the S-100 bus.

Things are a bit more complicated for 16 bit memory read or write. We must read/write the lower (A0=0) 8 bits of data, raise A0, and then read/write the odd 8 bits of data. It is necessary to completely remake the ALE and R/W strobes for the IBM bus the second time. The low order R/W cycle for the (A0=0) is exactly the same as described above except now when A2 clears the wait state setup by A1, the S-100 bus is still held in a permanent wait state by the not Q output of B1. This is because now the S-100 signal SIXTN went low triggering B1. After the low order address data is latched into the 8282 (see above) there is a short delay via B2 after which the falling edge of B2 triggers the start of the second cycle. Pins 1 and 15 of the 74LS368 (figure 4), which enable the read/write strobes, temporarily go high. They then return low. The PC sees this as a new R/W strobe. See figure 6 for a detailed timing diagram. We also raise A0 via single shot E and relatch the address lines by a pulse from single shot D. When the R/W strobe comes down, the IBM Boards think a new board access is required. After the (variable length) wait state is generated (via IBM WAIT) the single shot F is triggered. This in turn clears B1, which in turn removes the wait state held on the S-100 bus.

Both I/O read and write are assumed to be carried out as 8 bit transfers by the S-100 CPU. sINP generated inputs will arrive from the IBM bus (if the correct port is mapped, see below) to the "data in" S-100 bus lines via the 74LS245 shown in figure 3. A careful analysis of the diagram will show that pin 19 of this chip (the output enable pin), will always go low when sINP goes high and IBM SEL is high. sOUT generated outputs will travel to the IBM bus via the 74LS244(A) shown in figure 3. Pins 1 and 19 of this chip go low when sOUT and IBM SEL are high.

If you are using a CPU that has an 8088 (i.e. no 16 bit requests) you can simplify figure 3 by changing jumper b1-2 to position b2-3 and c1-3 to c1-2. This effectively removes the 8282 latch forcing all "data out" to go through the 74LS244(A) and all "data in" to arrive via the 74LS245.

Figure 4 shows how the read and write strobes are connected to the IBM bus. Because some static memory boards may not put a wait state on the IBM bus (ie. lower IO RDY) when they are read or written to, and the circuit in figure 2 requires at least a minute pulse of a wait state, a minimal IBM WAIT state signal is generated via single shot H.

The 74LS245 in figure 4 is a bidirectional transceiver for the 8 data lines going to the IBM bus. It is enabled when IOR,IOW,MEMR or MEMW goes low. Pin 1 determines the direction. You may jumper the IBM bus IRQ lines as you wish.

The 8284 clock generator circuit is exactly as utilized by IBM. It is necessary to place this circuit on the IBM motherboard to have sharp signals.

The only remaining section is the mapping system (figure 5). Because

one will normally only want a certain portion of the S-100 16M bytes of memory to map into the IBM bus hardware we must enable the board only when this occurs. The 74LS684 can be jumpered to go low only when the correct memory range is addressed. The jumpering is done so that a high or low address line on one side of the chip matches a high or low on the other. This should be familiar to anybody putting memory boards together. If the correct memory range is being addressed, IBM SEL will go high and be used as described above.

Mapping of I/O ports is essentially the same. Up to 4 blocks of I/O ports (figure 5) can be used. Clearly this could be expanded. For example the PC uses ports 3B0 to 3BF for its monochrome display. The jumpers shown are addressed for this range. Note that unconnected inputs on a 74LS682 are internally pulled high.

Table I shows the capacitor and resistor values for the one-shots used in figures 2 and 4. While putting the board together I used variable resistors to fine tune the system. There is much room for upward variation in the values. Figure 6 shows the scope traces at various points on the board when the small test program shown in figure 7 is run. This test can be used when adjusting the R-C values of the one shots. In the program a continuous series of 16 bit, 8 bit even and 8 bit odd reads to the PC video board are made. The scope is triggered on pin 12 of the 74LS11 in figure 2. The scan rate is 1 micro-sec/cm.

Table 1

Resistor-Capacitor Values
For 74LS123's

IC	Resistor	Capacitor
A1	9Kohms	510pF
A2	14Kohms	50pF
B1	24Kohms	510pF
B2	18.8Kohms	47pF
C	10.7Kohms	56pF
D	6.1Kohms	56pF
E	16.3Kohms	330pF
F	5.8Kohms	30pF
H	1Kohm	180pF

This is not a simple project. However the advantage of having the ability to use PC boards in your S-100 system makes this project worthwhile. There are many ways this board could be improved. I intend, at a later date, to upgrade it to be compatible with the IBM-AT bus system, and to an on-board DMA controller on the motherboard. I will be pleased to collect any

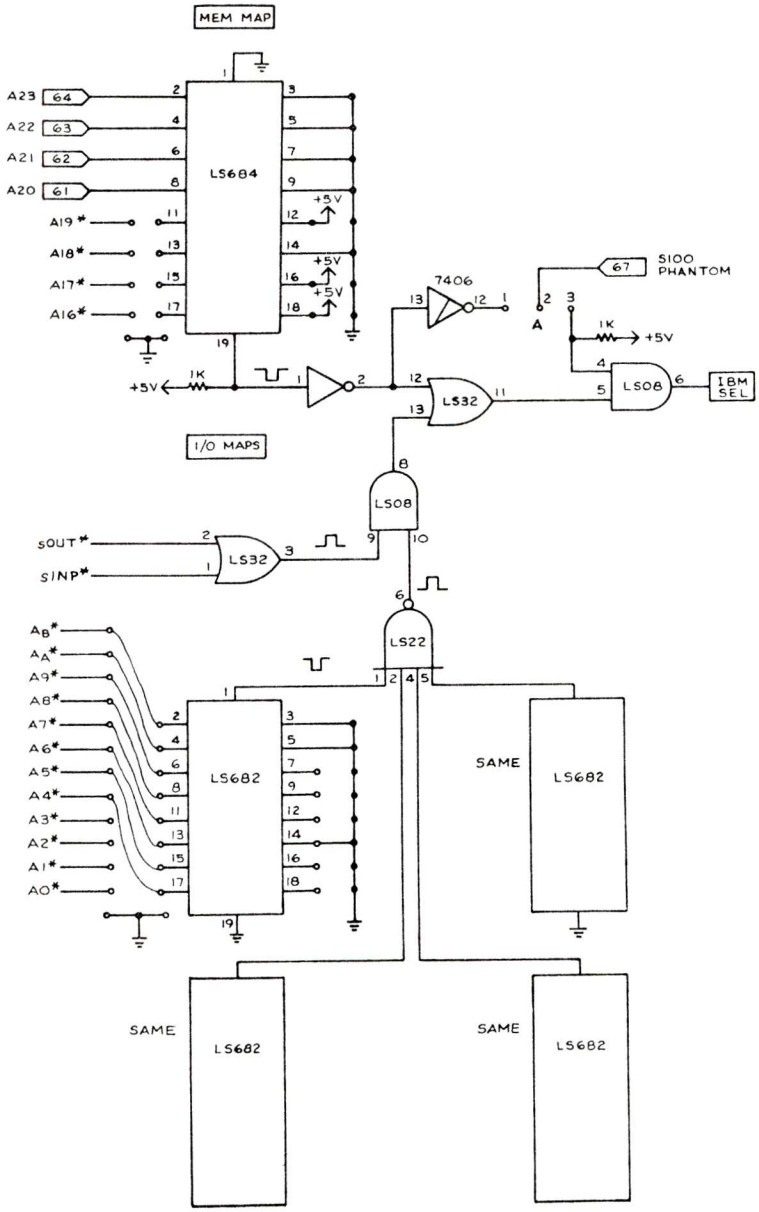


FIGURE 5

circuit improvements you may have.

Finally, it must be pointed out that to utilize the full advantage of this system you must interface it with the correct software. For example, I use IBM's ROM BIOS drivers for their monochrome and color video boards almost word for word out of their technical manual. These form part of a CONIO device driver for my custom MSDOS system.

In future articles I hope to tell you about my S-100/PC keyboard adaptor unit and an S-100 8086 slave CPU (which utilizes an 8087 and 8089) that has the best elements of many commercial CPU boards. These boards when combined with good software (written by a friend of mine that for example utilizes the 8089) yields an S-100 system that today knocks the socks off anything in the non S-100 arena.

```
;Program to generate test pattern
;for IBM-PC converter board. Code
;written for Digital Research's
;ASM86.
          CSEG
          org    0
          mov    ax,0b000h
          mov    ds,ax
          mov    bx,0
;
more:     mov    Word Ptr [BX],ax
          mov    Byte Ptr [BX],al
          inc    bx
          mov    Byte Ptr [BX],al
          dec    bx
          jmps   more

          DSEG
          org    100h
          db     0
          END
```

FIGURE 7

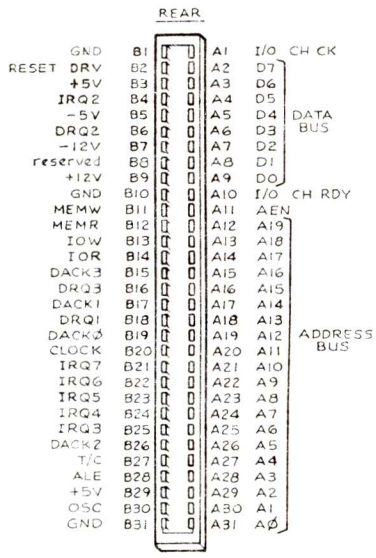


FIGURE 8 — BUS PIN FUNCTIONS

John Monahan is a molecular biologist. He received a Phd in Bio-Chemistry from MacMaster University in Ontario Canada. John has been a computer hobbyist for the past ten years and has built several homebrew systems. He has been a member of the Amateur Computer Group of New Jersey for over 9 years and recently moved to the San Francisco bay area. He can be contacted at: Box 1908, Orinda CA 94563.

Running MS-DOS On S-100 Systems

It should be pointed out that the IBM-PC version of MS-DOS, as well as the versions for most of the PC clones, can be purchased separately of the systems at most computer stores. However, these versions contain system I/O drivers that are not compatible with most S-100 systems. Neither Microsoft nor any of the equipment manufacturers provide the system initializing files necessary to port MS-DOS to another hardware configuration. Seattle Computer Products, did, until a short time ago sell a version of MS-DOS that contained the SYSINT files necessary for the job.

If you own a CompuPro system then you can buy a version of MS-DOS 2.0 already configured for your system from Computer House, 20 Oak Grove Ave, Woodacre CA 94973, tel: (415)897-6387.

If you own a Lomas Data System, then you can buy a preconfigured version of MS-DOS from them (see their ad in this issue).

People using the Tarbell Electronics 816 S-100 board can boot a standard IBM-PC version of MS-DOS. Although they would be better off buying a non-IBM version of MS-DOS which contains disk versions of Basic and BasicA.

Concurrent-DOS from Digital Research is also MS-DOS compatible. The current version (3) is compatible with MS-DOS Version 1. DRI is expected to shortly release Concurrent-DOS 4.1 which is expected to be MS-DOS Version 2 compatible (and also is expected to include the GEM Macintosh-like user interface). Concurrent-DOS V3 is available for several S-100 systems. For example, CompuPro furnishes an implementation for use with their S-100 80286 CPU and new PC-compatible video display cards (a review of which is in the works). And they are promising to furnish Concurrent-DOS V4 as soon as it is available.

DRI, like Microsoft, furnishes Concurrent-DOS strickly as an OEM product. It does sell, via retailers, an IBM-PC (and compatibles) version. However, the software necessary to configure it for a particular hardware configuration is provided only to OEMs. DRI no longer markets any configurable operating systems to non-OEM customers and no longer provides support for its old configurable CP/M operating systems.

We are attempting to find a way in which other S-100 8086 users can implement MS-DOS or Concurrent-DOS on their systems. If any reader can help in this regard please call me.

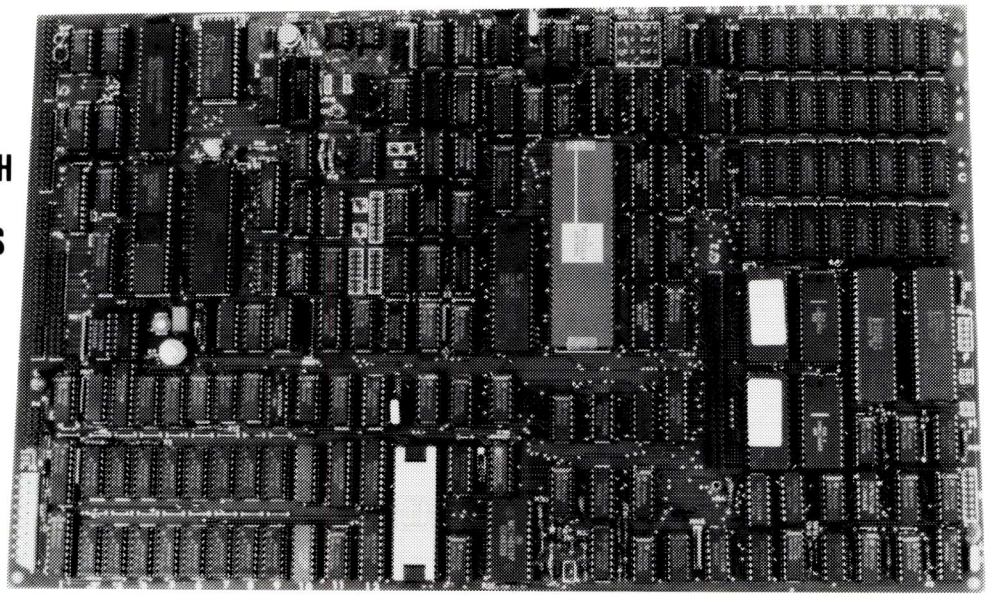
Sol Libes, editor

FOR THE **BEST** OF US . . .

THE CYPHER™

A COMPLETE 68000 & Z 80 SINGLE BOARD COMPUTER SYSTEM WITH ULTRA-HIGH-RES GRAPHICS!!

FREE 68000 FORTH AND CYPHER-DOS



FREE 68000 FORTH AND CYPHER-DOS

- 68000 & Z80 DUAL PROCESSORS (BEST OF BOTH WORLDS!).
- 256K to 1 MEGABYTE MEMORY. (4164 OR 41256 DRAM).
- DOUBLE DENSITY FLOPPY DISK CONTROLLER (8", 5¼" OR 3½" WD 2793).
- DMA CONTROLLER FOR FAST IMAGE TRANSFERS TO/FROM VIDEO MEMORY. (INT 8237).
- 2 RS232 SERIAL PORTS (ZSIO).
- 24 BIT ADDRESS MANAGEMENT FOR Z80.
- 4 LAYER P.C.B. (9¼" x 14¾").
- RUNS CP-M-80 2.2, CP-M-80 3.0, CP-M-68K
- ULTRA HIGH RESOLUTION GRAPHICS, 128K, PROGRAMMABLE UP TO 1024 x 1024 RESOLUTION (NEC 7220, GREAT FOR CAD SYSTEMS!).
- REAL TIME CLOCK (MULTITASKING CAPABILITY!).
- TWO CHANNELS OF D/A AND A/D, 12 BIT RESOLUTION (MUSIC! ROBOTICS! LAB WORK!).
- 16K TO 64K MONITOR EPROM.
- 4K TO 64K STATIC RAM.
- PROGRAMMABLE BAUD RATE GENERATOR.
- PARALLEL ASCII KEYBOARD INPUT.
- FULL 68000 EXPANSION BUS (60 PIN HEADER, BUFFERED).

- MANUAL **$20.00**
- BARE BOARD, EPROMS, Z80 BIOS, 68000 BIOS, Z80 MONITOR, 68000 MONITOR AND CYPHER DOS **$399.95**
- MINIMUM OPTIONED "CYPHER" WITH 68000/Z80, SERIAL I/O, 128K DRAM, 4K S RAM AND DISK CONTROLLER, ASSEMBLED AND TESTED **$1,099.95**
  PARTS KIT **$999.95**

* EXPANDED MEMORY OPTIONS – INQUIRE *

- COMPLETE "CYPHER" WITH 256 K DRAM, 128 K VIDEO DRAM, NEC 7220, REAL TIME CLOCK, A/D D/A DISC CONTROLLER, SERIAL I/O, ASSEMBLED AND TESTED **$1,499.95**
  PARTS KIT **$1,379.95**
- KEYBOARD **$94.95**
- SWITCHING POWER SUPPLY **$154.95**
- CASE **$94.95**
- HARD DISC HOST INTERFACE PLUG-IN CARD **$150.00**

VISA

master charge

SHIPPING CHARGES:
ALL PRICES ARE IN U.S. DOLLARS
SHIPPING: 8% SHIPPING, EXCESS WILL BE REFUNDED

PRICES SUBJECT TO CHANGE WITHOUT NOTICE.
CP-M IS A TRADEMARK OF DIGITAL RESEARCH.

MOTOROLA INTEL

MOTEL COMPUTERS LIMITED
174 BETTY ANN DRIVE, WILLOWDALE,
TORONTO, ONTARIO, CANADA M2N 1X6
(416) 221-2340

Interfacing To MS-DOS

by William G. Wong

Editor's Note: MS-DOS is today far and away the most popular single-user 16-bit disk operating system. Not only does it run on the IBM-PC but an increasing number of other systems (including 8086, 80186 and 80286-based S-100 systems) also use it. Also, there are other operating systems (e.g. Concurrent DOS from Digital Research Inc.,) that are compatible with MS-DOS. Therefore, I feel that our readers would like to know more about writing applications software to interface to MS-DOS and other MS-DOS compatible operating systems. It was with this in mind that we commissioned Bill Wong to write this series on software interfacing to MS-DOS.

Part I — the program segment prefix, access to DOS, and how to terminate a program

MS-DOS, from Microsoft, Inc., has become one of the most popular operating systems for 8086-based microcomputers. This is primarily due to the popularity of the IBM PC implementation called PC-DOS. This series describes application interface to MS-DOS V2.x and PC-DOS V2.x which will be referred to as DOS. DOS provides a standard interface to the disk file system and peripherals so that the same programs can be run on different 8086 machines running DOS, without change.

This series of articles will address the assembly language interface and how the functions are used by application programs. This first part covers the program segment prefix, access to DOS, and how to terminate a program.

Subsequent parts will cover character output, DOS CP/M-like file access and DOS UNIX-like file access. All numeric values listed in this article are hexadecimal unless noted otherwise.

Program Initialization and the Program Segment Prefix

DOS programs come in two flavors, .COM files and .EXE files. The primary difference between the two is that .COM files contain no relocation information and the program size is therefore limited to 64 kbytes. .EXE files contain relocation information which allow programs larger than 64 kbytes to be loaded.

In either case, the programs are loaded just above a memory structure called the Program Segment Prefix (PSP). The PSP is 256 bytes long and contains the following information.

Byte Offset	Description
0	INT 20 instruction (Terminate program)
2	First paragraph of unallocated memory
4	Reserved
5	Intrasegment (Short CALL) DOS entry point (Function code passed in register CL) (Valid only for functions 0-24)
6	Number of bytes in this segment
A	Termination interrupt vector (IP, CS)
E	Control break interrupt vector (IP, CS)
12	Critical error interrupt vector (IP, CS)
16	Reserved
2C	Program Environment segment number
2E	Reserved
50	Intersegment (Long CALl) entry point to DOS (Function code passed in register AH) (Valid for all function numbers)
5C	Formatted parameter area 1
6C	Formatted parameter area 2
80	Default disk transfer area (128 bytes)

The PSP looks very much like the base page of a CP/M-80 system. A call or jump to offset 0 will terminate the program and the operating sytem can be accessed at offset 5 although only the CP/M-style DOS functions (0-24) are accessible through this interface. Also, the default disk transfer area and formatted file name parameters are at their CP/M locations. This similarity aids in translation of programs from CP/M to DOS.

The Program Environment segment contains ASCII strings of the form "NAME=parameter". This is one method of passing parameters to applications and is set via program control or a DOS command (SET, PATH, or PROMPT).

The default disk transfer area contains the unformatted parameter string (normally the starting command line) less any redirection characters and file names. The first byte of this area contain the size of the parameter string followed by the string itself. This string is formatted into the two buffer areas at 5C and 6C. The initial value in the AX register indicates the status of these buffers as follows:

AL = 00 indicates first formatted drive specifier is valid
FF indicates drive specified is invalid

Move over, Crosstalk ™ . . .

The NightOwl's in town and he's packing a 16-bit MEX!

Last year, the NightOwl delivered MEX, the **M**odem **EX**ecutive that tamed the 8-bit communications frontier.

This year, he's doubled his byte with MEX-PC — the supercharged 16-bit communications package for the IBM-PC — and he's looking to take on the big boys, feature for feature.

$59.95 plus $5 for shipping and handling
(includes MEX-PC software and complete manual)

Supports all popular modems • Programmable for unattended operation • Extensive HELP overlay • Auto-dial and redial • Alternate long distance dialing (ALD) • "List" dialing with automatic baud switching • Instant defining of IBM-PC function keys • Fast creation of custom "smart" phone directories • All popular protocols — extended Christensen XMODEM (Checksum *and* CRC) CompuServe *A*, ASCII (X-on, X-off) odd-even-none bit parity • A CLONE routine for unlimited creation of customized versions • Full access to your own operating system and software while logged onto a host system • Delay-adjustable Break key • DOS-compatible commands • Supports all monitors, port switching, named directories, on-line printing • IBM-PC-XT-AT — all DOS levels • 110 to 19,200 baud on most equipment • Source code for any overlay available

"Individually, each of these features enhances the experience of telecomputing, but together they add up to enormous power and flexibility . . . one of the most innovative and sophisticated communications packages available . . . MEX has been greeted with universal acclaim."
That's how Link-Up magazine described the 8080 version of MEX last September. Now, there's MEX-PC!

You've struggled with overpriced, so-called smart terminal software long enough.

Now, experience the genius, the economy, the power! of MEX-PC.

VISA master charge

Give us a call at 1-800-NITEOWL

(in Wisconsin, call 414-563-4013)

NightOwl© software, inc.

Crosstalk XVI is a trademark of Microstuf, Inc., Atlanta, GA
MEX-PC is a trademark of NightOwl Software, Inc., Rt. 1, Box 7, Fort Atkinson, WI 53538

AH = 00 indicates second formatted drive specifier is valid
FF indicates drive specified is invalid

The other register settings depend upon the program type. COM programs have all segment registers set to the segment containing the Program Segment Prefix. The instruction pointer register (IP) is set to 100 hex where the program code resides. The stack pointer register (SP) is set to the end of the segment and 100 hex bytes are allocated to the segment. A zero is placed on top of the stack thereby allowing a program to exit via a short return instruction (RET) as many CP/M programs do (this is also a CP/M translation assist). Any COM programs which invoke another program must first free all unnecessary memory before invoking the program.

The EXE programs have more information than the COM files. The code segment (CS) and stack segment (SS) registers are set based upon this information as are the IP and SP registers. The data (DS) and extra (ES) segment registers are set to the Program Segment Prefix.

DOS Interrupts

DOS is accessed via 8086 interrupt instructions. An interrupt instruction (INT) has a single byte value (0-255) associated with it. This value is the interrupt number. An interrupt instruction indexes the interrupt vectors located at the base of memory. The vectors contain the new code segment (CS) and instruction pointer (IP) registers for the interrupt support routine. The interrupt instruction also saves the current FLAGS, CS and IP registers on the stack. The interrupt routines typically return control to the program using the interrupt return (IRET) instruction. However, some interrupt routines exit via long return (RET) instructions leaving the FLAGS register on the stack. Also, routines which cause program termination never return to the calling program.

Some interrupt vectors are predefined for 8086 error and hardware support functions. Others are allocated to DOS with the remaining vectors available for various functions, and some are reserved for future use. The DOS related functions are described in the rest of this section.

INT 20 (Terminate program and release all memory being used)

The active program is terminated. All file buffers are flushed but all files should be closed before the interrupt is issued. Interrupt vectors for INT 22 , 23 and 24 (termination handler, control-break handler, and critical error handler) are restored.

INT 22 is issued to terminate the process. The INT 22 vector from parent program will be the one restored and used. This is normally COMMAND.COM if only one other program is active.

The code segment (CS) register must hold the segment address of the Program Segment Prefix for proper termination.

INT 21 (DOS Function Request)

DOS functions are called using this interrupt. The function number is placed in register CL with other parameters being passed in other registers depending upon the function. These functions will be discussed in this and subsequent articles. In general, a DOS function request looks like:

```
MOV CL,DOS FUNCTION     ; cl := DOS function number
INT 21H                 ; perform DOS function
```

INT 22 (Issued when program is terminated)

INT 22 is invoked by DOS when a program is terminated. These vectors are initially set by the command processor, COMMAND.COM. Control returns to the command processor when an application terminates via this interrupt routine. Programs which start other programs can also regain control upon termination of the other program in the same fashion.

A program must set the values of this interrupt vector before initiating another program. The vector values are restored to their prior value when the initiating program terminates so it is not possible for a program to to process its own termination request.

INT 23 (Control-break intercept)

The control-break exit routine is called by DOS I/O routines if a control-break is issued from the keyboard. The program will continue execution if this routine exits via an interrupt return (IRET). The other way to exit from this routine is via a long return. This leaves the flags on the stack. The program is continued as with an IRET if the carry flag is not set. Otherwise, the program is terminated. All registers must be preserved if the program is to be continued.

INT 24 (Critical Error Handler)

The critical error handler routine is called whenever a critical I/O error occurs from DOS which is called via INT 21. Critical errors encountered from INT 25 or INT 26 are not handled by this error handler. Note, DOS performs 5 retries before issuing an INT 24. Exit from this routine is via an IRET. The new program state is included in the AL register as follows:

0 means that error should be ignored and program should continue execution.

1 indicates that operation should be performed again and program exeuction should continue. Note: retry may generate another error which would again cause an INT 24.

2 causes program termination via INT 23.

Upon entry DP:SI contains the Device Header Control Block address. The bits in register AH indicate the device type and error if the device is a block device. The following table contains the bit designations.

Bit	Description
0	Operation (0 = read, 1 = write)
1-2	Disk area affected: 1 = file allocation table area (FAT) 2 = DOS directory 3 = DOS data file area

The least significant byte of DI contains the error code with more detailed information. The following table contains the error code values.

Error Code	Description
0	Attempt to write to a write-protected disk
1	Unknown unit
2	Block device drive not ready
3	Unknown or illegal command
4	Block device data (CRC) error
5	Incorrect request structure length
6	Block device seek error
7	Block device unknown media type
8	Block device sector not found
9	Character device (printer) out of paper
A	Write fault
B	Read fault
C	General error not defined above

The stack contains the following information from DOS and the user program. Note, the first item is on top of the stack.

Register	Description
IP	DOS registers from INT 24
CS	
FLAGS	

AX	User registers at time of INT 21
BX	
CX	
DX	
SI	
DI	
BP	
DS	
ES	
IP	User registers from INT 21
CS	
FLAGS	

Interrupts are disabled when this routine is entered. The contents of registers SS, SP, DS, ES, BX, CX and DX must be preserved. Only DOS function calls 1 through 12 may be used as all others change the contents of the DOS stack which is currently being used by this interrupt support routine. Also, the DOS state may change such that the system will crash.

Although the contents of the device header are available, it should not be altered in any way. Control can be passed directly to the user program by restoring the user registers and executing an IRET instruction.

INT 25 and INT 26 (Absolute disk read and write)

These interrupts perform disk operations that bypass the normal DOS file system. Direct access to any sector on the disk is possible. All errors are returned directly to the calling program. Parameters are passed in the following registers.

Register	Description
AL	Drive number (A = 0, B = 1, etc.)
CX	Number of sectors to read or write
DX	First logical sector to transfer
DS:BX	Transfer address

Sector size is the physical sector size of the device. The first physical sector on the first cylinder is logical sector 0. Although entry to these functions is via an INT instruction, the functions exit via a long return (RET) instruction. This leaves the original FLAGS on top of the stack. These must be removed when the calling program regains control. The carry flag (not the one in FLAGS on the stack) will be zero if no errors occurred and it will be one if an error occurred. Error results are returned in AX. The error codes in AL match the ones returned in the DI register for INT 24. One of the following error codes will be returned in AH.

Error Code	Description
80	Attachment failed to respond
40	Seek failed
20	Hardware controller failed
10	Bad data (CRC) error
08	Data lost due to DMA overrun
04	Cannot find requested sector
03	Write attempted to write-protected disk
02	Address mark not found
00	Error not listed above

INT 27 (Terminate program but remain resident)

A program can terminate itself but remain resident in memory using this interrupt. This operation is normally performed by hardware or software interrupt support programs after they have initialized the corresponding hardware and interrupt vectors. Programs of this type remain resident and can be accessed by other programs which are run after the resident programs terminate. Access is typically performed via INT instructions and interrupt vectors which may point to table addresses.

A program may also terminate itself and remain resident using DOS function number 31. The advantage of using function 31 is that a completion code can be returned while none can be returned via INT 27. Also, the resident size of the program is not restricted to 64 kbytes.

There are also some restrictions on the program being terminated. First, the program may not be an EXE program which will be loaded in high memory. Second, the maximum size of the memory segment which remains resident is 64 kbytes. One time initialization code may extend past this point since it may be overwrittten by subsequent programs. Third, vectors for INT 22, 23, and 24 are restored when the program is terminated even if the program changed their values.

Upon exit, the program must have set the code segment registers to the base segment to be retained and the size of the segment in bytes must be placed in the DX register. The resident program is then considered an extension to DOS. The following is an example of terminating a resident program.

```
MOV DX,RESIDENT_SIZE      ; DX := resident size in bytes
INT 27H                   ; exit and remain resident
```

DOS Function Invocation

Unlike CP/M, DOS functions can be invoked through an number of different methods. The normal method is using INT 21 with the function number in the CL register. A short call to offset 5 in the Program Segment Prefix will work too but only for function numbers between 0 and 24. A long call to offset 50 will work for all functions but the function number is passed in register AH.

All other register values are function specific.

DOS Program Terminate Functions

The historical program termination has been through DOS function 0 or by a direct BIOS call. The later corresponds to INT 20 described earlier although INT 20 is not used as the BIOS entry point. DOS function does terminate a program which essentially performs an INT 20. All requirements and restrictions are the same. An example of this type of termination is:

```
MOV CL,0                  ; CL := terminate program function
INT 21H                   ; execute DOS function
```

A program may terminate but remain resident by using DOS function 31. This is similar to INT 27 but allows the resident portion of a program to exceed 64 kbytes and also return an exit code to the system. The DX register must contain the number of paragraphs (16 bytes/paragraph) to retain while the AL register contains the exit code.

This function DOES NOT release any other allocated memory which the program is using so all unnecessary memory should be released before issuing this function call. Here is an example of a resident program DOS terminate function call:

```
MOV DX,RESIDENT_PARAGRAPHS; dx := resident size/16
MOV AL,EXIT_CODE          ; al := program exit code
MOV CL,31H                ; cl := terminate resident program
INT 21H                   ; perform DOS function
```

There is also a DOS function which allows a program to terminate and still return an exit code. This is DOS function 4CH. The exit code is returned in AL. All files are closed but it is good practice to close the files before invoking this function. This function is the only way to terminate a program without having to set the code segment register (CS) to the segment containing the Program Segment Prefix. An example of this function is:

```
MOV AL,EXIT_CODE          ; al := exit code from program
MOV CL,4CH                ; cl := terminate program function
INT 21H                   ; execute DOS function
```

Summary

Although program termination, interrupts and the Program Segment Prefix represent a small part of the DOS interface, it is a start and does allow creation of a minimal program. The next article will address console character input and output which will enable you to write a program that prints something.

Roll Your Own PC Clone

By Sol Libes

How I assembled a PC clone for under $1,000.

For some time now I have been aware of PC clone boards being sold via mail order and at computer flea markets. I was very dubious about them, having heard about all sorts of compatibility problems with them (see Henry Kee's article in last month's *Micro/Systems Journal*). However, I finally decided to take the plunge.

It happened at a computer flea market I recently attended here in New Jersey. Several vendors were selling parts that ranged from bare motherboards, to boards with a bag of parts, to a board with sockets, on up to a fully populated and tested board which came with a BIOS ROM and a 90-day warranty. The bare-board price, with some rather skimpy instructions (a board layout diagram, parts list, and if you were lucky, a schematic diagram) cost $60, while the populated/tested/guaranteed board typically cost $295. I decided to go for the latter. One vendor even offered me a board with a copy of the IBM-PC BIOS ROM. However, being in the public position that I am in, I ended up buying a board with the board manufacturer's BIOS ROM.

What Did It All Cost?

The motherboard cost me $295. A 100 Watt power supply was $95 (a 135W supply was about $110). A keyboard was $75, a graphics controller card $130, and a floppy disk controller card, with cable, was $75. I also bought a used half-height double-sided floppy disk drive for $50 (new they were $85) and a monochrome display for $65. I then bought a copy of MS-DOS from a Compaq dealer (list was $60 and I got it for $50). A total cost of $835.

A few weeks later I decided to get a case for the unit ($70), another floppy, and additional memory chips (192K for $30) to bring the total memory up to 256K. With these last items the price came to $985. I have also ordered a 10.5Mbyte drive and controller for the unit ($450) and the dealer swapped my 100W power supply for a 135W unit for the difference ($15). This brought the entire system price up to $1,475. Not bad for a complete 10.5Mbyte PC compatible system. I figure I saved about $1,200 over a PC compatible (e.g. Compaq) and about $1,700 over an IBM/XT.

It even looks like an IBM/XT.

Putting Together A Taiwan Copy

The motherboard, disk and display controller cards and the keyboard, power supply and cabinet were made by a company called "Super Computer". I think they should change their name to "Not So Super Computer"!

The motherboard came with a few xeroxed sheets on how to turn on an assembled unit and how to open the case to install additional boards... that was it. There was no documentation on how to connect the power supply or how to set the switches on the motherboard. Connecting the power supply presented a problem since there are two identical connectors from the power supply to be plugged into the motherboard. What I ended up doing was looking at an IBM-PC to see how the connections were made and copying it.

I connected the power supply to the motherboard and to my drive. Plugged in the display and disk controller cards and the keyboard to the connector on the back of motherboard. Then I hooked up my display monitor to the display controller card and put the Compaq DOS system disk in drive A. There were no instructions as to which was drive A so I used an IBM-PC as my guide. *The first rule with this system is — when in doubt about what to do, try copying your nearest IBM-PC.*

I switched on power and lo and behold a cursor appeared on the screen, then a message "testing memory" followed by some noises from the A drive followed by the MS-DOS opening messages. If I discount the unpacking time and time checking with an IBM-PC to see how to connect things then it took me only about 10 minutes to assemble a working system. When I think back as to how long it took me to get my first S-100 CP/M system running, this was an absolutely pleasurable experience. I would put the assembly of the system almost at the level of assembling a hi-fi system.

Here are some tips, if you decide to

follow in my footsteps. Drive A is the drive at the end of the cable (the cable contains the drive addressing rather than the select jumpers on the drive). Also, drive-A should have the terminating resistor pack. Be sure there is no resistor pack in drive B.

Although the system ran immediately there were some problems. The display was 40 columns x 25 lines and I could not access drive B, when it was installed. The problems were caused by incorrect switch settings on the motherboard. Here again I used an IBM-PC as a reference. However, there are only half the number of switches on the Super Computer motherboard as there are on the IBM-PC. The PC has two groups of 8 switches (SW1 and SW2) while the Super has only one group (SW1). A little experimentation proved that the Super's SW1 was the same as the PC's SW1 and that SW2 is really not needed. SW2 on the IBM-PC tells the system how much memory is in the system. The super computer tests memory and finds that out itself, so no need for SW2. I set the switches as if it were a PC and the display switched to 80 x 25 and I was able to access both drives. The installation was complete!

The Motherboard

The motherboard appears to be built with about the same quality as IBM's. It has 8 plug-in sockets, which is the same as an XT and 3 more than a PC. Since the unit has 8 sockets, 135 Watt power supply (the PC has a 63 Watt supply while the XT has 135W unit) and the BIOS ROM contains a hard disk boot, the manufacturer calls the unit the "Super XT". If you add the hard disk then the unit is really a copy of an XT rather than a PC.

Physically, the unit looks very much like an IBM motherboard. There are the same number of ROM sockets and a socket for the 8087 math co-processor chip as well as a connector for a speaker.

What's Different?

When you look down inside the chassis you would swear you are looking at an IBM XT, that is how similar the unit is to the construction of the IBM-PC. However the case is a little larger in the depth allowing more room for routing the hard disk drive cables that are a real problem in an IBM box. No speaker was provided, but there is a mounting bracket and I intend (if I ever get a free minute) to run down to a Radio Shack store and pick up a miniature speaker.

There were no instructions on how to assemble the cabinet. It took me about two hours to assemble it and get all the components installed. When I was finished I had two brackets left over and quite a few nuts, screws and washers. Apparently they were supposed to go somewhere but I couldn't figure out where. Again, no assembly instructions were provided.

There are sockets for ROMs, just as in the PC. However, there is only one ROM in the unit, the BIOS ROM. There are no Basic ROMs as in the PC. This means that the Basic interpreters (there are two) must reside on the disk. Therefore, the IBM versions of Microsoft Basic will not execute on this system since they use some of the IBM ROM Cassette Basic routines. The solution is to obtain a copy of the Microsoft version of disk basic (called GWBasic). The Compaq version of MS-DOS included the disk resident versions of Basic.

Another thing that is different is that the BIOS memory test takes about 2 seconds compared with upwards of 30 seconds on the PC (more memory, more time).

The Keyboard

The organization of the keyboard is identical to that of the IBM-PC. It also included three LEDS that are lacking on the PC. There are LED indicators for power, caps lock, and number lock. This is a really neat feature. However, they glow so dimly that they are hard to tell whether they are on or off.

The keyboard has a very different feel from that of the IBM. Although the

feel is acceptable I find that of the PC preferable. The PC's keys have more resistance and you have to press harder while the Super's keys require very light pressure to make contact.

There is also a little relay in the keyboard that makes clicking sounds every time a key is pressed or held down to repeat. This was probably done because the keys are very quiet compared to the IBM's noisy keys. Fortunately, there is a control on the underside of the keyboard that allows one to turn off the clicking.

Compatibility

How compatible is the unit to that of the IBM-PC? There are two software packages that the industry generally uses to test PC clone compatiblity... Lotus 1-2-3 and Microsoft's Flight Simulator game. I can report that this units runs both without any problems. I also ran GWBasic with some rather large Basic programs that I have written and WordStar and can report no problems with these either. If anything they seem to run a little better on this system than on the IBM-PC. First of all there is little of the screen flicker that is most annoying on the IBM-PC. And second, the half height floppy drives are much quieter and have a little faster read/write times.

Conclusions

All in all, I think I made a wise investment. For an investment of about 2-3 hours to assemble the system I saved about $1200 on the dual floppy system and well over $2,000 on the hard disk system.

Board Sources

Here are some mail order sources for foreign PC-compatible motherboards, plug-in cards, keyboards and power supplies. Check them out carefully as many are small outfits.

American Digital Discount
4040 Spencer St
Torrance Commerce Center
Torrance CA 90503
(213)542-3578

Computer Supply
Box 164
Valley Stream NY 11582
(516)239-1855

J C Computer Inc
9238 Katella Ave
Anahiem CA 92804
(714)821-8922/23

JDR Microdevices
1224 S Bascom Ave
San Jose CA 95128
(800)538-5000

Lanton Computer Systems
37 Juneau Blvd
Woodbury NY 11797
(800)645-4441

Mendelson Electronic Surplus
340 East First St
Dayton OH 45402
(513)461-3525

Micro Data Supplies
22295 Euclid Ave
Cleveland OH 44117
(800)321-3552

Microware Exceltak
421 Grand Ave
So San Francisco CA 94080
(415)952-5375

Super Price Inc
25108 Margurite Parkway
Mission Viejo CA 92691
(714)730-9336

The Great Salt Lake Computer Co Inc
1780 W 2300 South
Salt Lake City UT 84119
(800)545-2633

United Computer Resources
931 Arch St
Philadelphia PA 19107
(215)849-0589

LOMAS DATA PRODUCTS INVITES YOU TO:

SHARE THE THUNDER.

The S100-PC-TM offers the following standard features:

- ☐ High performance THUNDER186 8Mhz 80186 processor
- ☐ 512K bytes of RAM (expandable to 1Mbyte)
- ☐ 4 serial ports to support up to four users
- ☐ 3 Centronics compatible parallel ports
- ☐ Concurrent DOS operating system allows execution of both CP/M-86 and MS-DOS (PC-DOS) programs
- ☐ 5¼" IBM-PC compatible floppy drive
- ☐ 40 Mbyte high performance Winchester drive
- ☐ Attractive 10 slot desktop enclosure

In addition, a number of options are available including: larger Winchester drives, more user ports, 80286 processor, graphics support and additional operating systems (MS-DOS and CP/M-86).

S100 BUS boards products & support for the system integrator . . .

All of LDP boards are fully tested to exacting standards and carry a one year warranty. We specialize in 16-bit products & support the major operating systems for 16-bit processors: CP/M-86*, CONCURRENT CP/M-86*, and MS-DOS (PC-DOS).

■ **THUNDER186 — THE ONLY COMPLETE S100 BUS, 16 BIT SINGLE BOARD COMPUTER AVAILABLE TODAY.**
Concurrent CP/M-86, which in addition to running CP/M-86 programs, runs MS-DOS programs. Comes complete, ready to plug into an enclosure and run. 256K bytes of RAM only . . . . . **PRICE $1595.00**

■ **LIGHTNING ONE***8086/8088 CPU**
8086 or 8088, with 8087 and 8089 coprocessors. Up to 10 MHZ operation . . . . . **PRICES start at $425.00**

■ **HAZITALL SYSTEM SUPPORT BOARD**
2 serial, 2 parallel ports, battery protected clock calendar. Hard disk controller host interface . . . . . **PRICE $325.00**

■ **LDP 128/256K DYNAMIC RAM**
Advanced dynamic RAM with LSI controller for failsafe operation, parity . . **PRICE** 128K—**$395.00,** 256K**$649.00**

■ **RAM67 HIGH PERFORMANCE STATIC RAM**
High speed (100ns) low power CMOS static RAM. 128K bytes, extended addressing . . . . . **PRICE $995.00**

■ **LDP72 FLOPPY DISK CONTROLLER**
Single/double density, single/double sided disks, both 8" and 5¼" inch drives simultaneously . . . . . **PRICE $275.00**

■ **LIGHTNING 286—80286 CPU BOARD**
Offers 4 times the performance of a 5MHZ 8086 CPU while maintaining software compatibility . . . . . **PRICE $1095.00**

■ **OCTAPORT 8 PORT SERIAL BOARD**
0 to 19200 baud operation real time clock interrupt. Ideal for multi-user systems such as MP/M-86* **PRICE $395.00**

S100-PC-TM is a trademark of Lomas Data Products, Inc.
**CP/M-86, MP/M-86 and CONCURRENT CP/M-86 are trademarks of Digital Research. **MS-DOS is trademark of Microsoft.*
****Lightning One is trademark of Lomas Data Products, Inc.*

S100-PC-TM: The LDP Multi-user S100 Bus System offers high performance at a "low" price . . . plus, "our" system is expandable and upgradeable!

PRICE An unbelievable **$6995.00**

Call today!

LDP

LOMAS DATA PRODUCTS, INC.
66 Hopkinton road, Westboro, MA 01581
Tel: (617) 366-6434 ☐ Telex: 4996272

Dealer inquiries invited.

For orders outside the U.S., contact our exclusive dealers: ☐ **Australia** — LAMRON PTY. LTD., (02) 85-6228
☐ **Malaysia** — EXA COMPUTER (M) SENDIRIAN BERHAD, 795284

Super assemblers plus the world's largest selection of cross assemblers!

All 2500AD Assemblers and Cross Assemblers support the following features:

POWERFUL LINKER

—Links up to 128 files
—Allows files to be used just for external reference resolution
—Separate code and data space
—Unlimited global and external symbols
—10 **significant** characters per symbol, no limit to length
—Submit or batch mode as well as command invocation, for easier linking of a large number of files

ASSEMBLER DIRECTIVES

Storage Control:
- ORG, ORIGIN
- END
- DB, DEFB, BYTE, STRING
- DW, DEFW, WORD
- LWORD, LONGW
- ASCII
- DS, DEFS, BLKB
- BLKW

Definition Control:
- EQU, EQUAL
- VAR, DEFL
- MACRO
- ENDM, MACEND
- MACEXIT
- EXTERN, EXTERNAL
- GLOBAL, PUBLIC
- ASK

Assembly Mode:
- RADIX
- DATA
- CODE
- MOD32 ON & OFF
- COMMENT
- INCLUDE
- DRIVES

Conditional Assembly:
- IFZ, IFNZ, COND
- IFTRUE, IFFALSE
- IFDEF, IFNDEF
- IFSAME, IFDIFF
- IFEXT, IFNEXT
- IFABS, IFREL
- IFMA, IFNMA
- ELSE
- ENDC, ENDIFT
- IFCLEAR

Listing Control:
- LIST ON/OFF
- MACLIST ON/OFF
- CONDLIST ON/OFF
- PASS1 ON/OFF
- PAGE, EJECT
- TITLE, HEADING
- SUBTITLE
- PW
- PL
- TOP

Additional Motorola Directives:
- FCC
- SET, SETDP
- PAG
- NAM
- STTL
- XDEF
- XREF
- FCB, FDB, RMB
- LONG

Run time commands (invoked while assembly is in progress):

^S—Alternately start and stop assembly
^C—Terminate assembly
^T—Display output at terminal
^P—Display output at printer
^D—Send output to disk
^B—Both terminal and printer or disk
^N—Turn off output display

1 Year of free support and updates now included!

You will receive a special support phone number on all products purchased after 3/1/85 allowing free updates and a full 12 months of support, included in the purchase price.

Features unique to these 2500AD products:

6800 FAMILY—"S"-record output option, special directives for dealing with page zero, absolute or relocatable modes.

68000—"S"-record output option, S-19, S-28 and S-37.

65XX FAMILY—Special directives for dealing with page zero.

Z-8—Register naming supported, TEK HEX output format.

8748—Register naming supported, INTEL HEX output.

8051/44—Register naming supported, INTEL HEX output.

8096—Register naming supported, INTEL HEX output, works for the 8097 as well. "Generic" calls and jumps allow assembler to determine long or short jumps.

Z-8000—Includes 8080/Z-80 to Z-8000 source code translator, uses the 2500AD syntax, not source compatible with Zilog. Includes powerful segmented linker.

Z-80—Includes an Intel 8080 to Zilog Z-80 source code converter. Includes the 2500AD linker, not compatible with Microsoft at the link level.

8086/88 & 80186—Includes an 8080/Z-80 to 8086 source code translator that will convert 8080/Z-80 source code to 8086/88 source code. Includes linker, not link compatible with Microsoft. Code, Data, Stack, and Extra segments supported.

	Z80 CP/M®	ZILOG SYSTEM 8000 UNIX	IBM PC MSDOS	IBM PC CP/M 86	OLIVETTI M-20 PCOS
Z8000™	$299.50	$750.00	$299.50	$299.50	$299.50
Z80	99.50	500.00	199.50	199.50	199.50
Z8	199.50	500.00	199.50	199.50	199.50
8086/88	199.50	750.00	99.50	99.50	199.50
80186	199.50	750.00	199.50	199.50	199.50
8748	199.50	500.00	199.50	199.50	199.50
8400/84C00	199.50	500.00	199.50	199.50	199.50
83C351	199.50	500.00	199.50	199.50	199.50
8044/51	199.50	500.00	199.50	199.50	199.50
8080	199.50	500.00	199.50	199.50	199.50
8085	199.50	500.00	199.50	199.50	199.50
8096	199.50	500.00	199.50	199.50	199.50
68020	399.50	750.00	399.50	399.50	399.50
68000,08,10	299.50	750.00	299.50	299.50	299.50
6800,02,08	199.50	500.00	199.50	199.50	199.50
6801,03	199.50	500.00	199.50	199.50	199.50
6804	199.50	500.00	199.50	199.50	199.50
6805	199.50	500.00	199.50	199.50	199.50
6809	199.50	500.00	199.50	199.50	199.50
68C11new	199.50	500.00	199.50	199.50	199.50
32000	399.50	750.00	399.50	399.50	399.50
COPS400	199.50	500.00	199.50	199.50	199.50
NSC800	199.50	500.00	199.50	199.50	199.50
6301	199.50	500.00	199.50	199.50	199.50
62801new	199.50	500.00	199.50	199.50	199.50
6501/11	199.50	500.00	199.50	199.50	199.50
6502	199.50	500.00	199.50	199.50	199.50
65C02	199.50	500.00	199.50	199.50	199.50
1802	199.50	500.00	199.50	199.50	199.50
F8/3870	199.50	500.00	199.50	199.50	199.50
NEC7500	199.50	500.00	199.50	199.50	199.50
NCR/32	399.50	750.00	399.50	399.50	399.50

Subtotal $ _____ $ _____ $ _____ $ _____ $ _____

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Phone _____
Make and model of computer system _____

☐ COD (2500AD pays COD charges)
☐ VISA or MasterCard
Number _____
Expiration Date _____

TO ORDER. **Simply circle the product or products you want in the price columns, and add up your order.**

Total $ _____

Check one:
☐ 8" Single
☐ Osborne
☐ IBM PC
☐ Cartridge tape
☐ Apple (Softcard)
☐ Kaypro DSDD
other formats available, please call!

Shipping UPS Blue Label no charge, $20.00 shipping for International per unit.
$ _____

Total Order $ _____

Signature _____ 224

2500ADSOFTWARE INC.

P.O. Box 4957, Englewood, CO 80155, (303) 790-2588 TELEX 752659/AD

ZCPR3
A CP/M-80 V2.2 Enhancement

by Randy Reitz

A brief history of ZCPR

ZCPR3 is the latest installment in the ZCPR series of CP/M 2.2 enhancements. ZCPR3 is the work of Richard Conn. He was one of the group of original developers of ZCPR. Back in 1980 ZCPR was put in to the public domain as a replacement for the CP/M Console Command Processor (CCP). CP/M 2.2 provided a way to divide up mass storage into areas identified by "user" numbers. You make use of this feature and enter a user area with the USER command. That's fine. The problem is now when you type a command, the CP/M 2.2 CCP looks only in the current user area of the disk for the command.COM file. This is incredibly dumb. ZCPR fixed this deficiency and added some more useful resident commands to the bargain (since it is written in Z80 assembly language, more code can be packed into the 2k-bytes of space alloted for the CCP). ZCPR became an instant success. I don't know of many CP/M 2.2 systems that don't have ZCPR running.

Richard Conn offered ZCPR2 sometime in 1981/1982 and improved on ZCPR slightly. ZCPR2 offered the option of naming the disk/user-number combinations so that they became even more convenient to use. ZCPR2 required more memory than the 2k-bytes available in the CCP so it required that the CP/M 2.2 system be moved down to make room available above the system. This sufficiently complicated the installation procedure so that many people decide not to convert since the benefit gained did not offset the pain of installation.

Now Richard is offering ZCPR3. Again, to his credit, ZCPR3 is in the public domain (for private use, not to be resold) via SIG/M. It can be found on RCP/M systems and in computer club software libraries. ZCPR3 retains

A CP/M-80 CCP replacement and how you go about installing it.

all the good features of its predecessors and adds considerably more features by way of a cooperating set of utilities and application development tools. ZCPR3 is also commercial. Echelon, Inc., 101 First Street, Los Altos, CA 94022 is marketing ZCPR3 and utility packages, documentation and some of Conn's other software (e.g., his SYSLIB3, VLIB, Z3LIB, and Discat packages. Echelon also sells a complete line of assembly language software development programs including relocating macro assembler with linker and librarian, five code translators, two debuggers, and disassembler.)

Echelon provides support to ZCPR3 users that purchase the software from them. They offer telephone consultation and a fortnighter newsletter. The newsletters (called Z-News) offer tips on ZCPR3 installation and usage as well as "8-bit versus 16-bit" editorials. For example, newsletter 004 contains a "Note about value" that concludes ZCPR3 is a real bargain since it cost only $.0001 per byte versus Unix at $.03 per byte. This information is useful for folks who buy software by the byte; but such logic makes me skip the editorials and hype and just read the technical tips. The tips are useful, especially the installation and utility usage tips.

What is ZCPR3?

ZCPR3 has gone beyond a simple CP/M 2.2 CCP replacement. By virtue of a reserved "message" area above the BIOS, ZCPR3 arranges to have its over 70 utilities interact with each other and the operating system to produce an environment that is greater than the sum of its parts. Many of the utilities are screen-oriented so that along with the menu capabilities of ZCPR3's shells, very friendly interfaces can be developed (as demonstrated by the utilities MU3, DU3, SHOW, MENU, VMENU and VFILER). The space above the BIOS is also used to hold the system segments — areas designated for special purpose programs such as an RCP (Resident Command Processor), NDR (Named Directory file)., FCP (Flow Control Processor) and an IOP (Input/Output Package). All these processors and the features they contain are optional and extensively optioned. A "full" implementation of ZCPR3 requires 5k-bytes of space above the BIOS. This sounds like a lot to take away from a 64k-byte system; but considering that only a few CP/M 2.2 application programs make use of all of the available space, it isn't much for the capability it brings.

8080/8085 users note that ZCPR3 can be assembled to run on an 8080 or 8085 processor. Fewer features can be packed in since the 8080/8085 code is not as compact as the Z80. Most of the utilities can be assembled for 8080/8085 or Z80 via an equate that permits assembly using either Z80 or 8080 instructions.

Documentation

I received a copy of the preliminary documentation with ZCPR3. The ZCPR3 "Sampler" is credited to Mr. Conn and the staff of Echelon. It contains two major sections, the ZCPR3 Installation Manual and examples called a User's Perspective.

The Installation Manual weighs in at 121 pages. I am sure I could describe just the ZCPR3 installation in a lot less than 121 pages; but this installation manual includes detailed descriptions of all available options. This situation is OK, even commendable, but the organization of the manual is poor. The ZCPR3 Newsletter 003 details an installation procedure that isn't as cluttered with detail. My reaction to the installation manual was that it tried to give you all the information on the numerous options before installation begins.

At least 3 hours of reading is required to get an appreciation of what was needed to install ZCPR3. I spent another 3 hours with the BIOS modifications and other installation activities. The critical BIOS modifications were presented by way of excerpts from a working BIOS. This requires the reader to understand what the excerpts are doing, ignore the extraneous code, symbols and decide how the excerpts should be incorporated into his own BIOS. Not an easy task!

The other section is called a "USER'S PERSPECTIVE". This document is by Mr. Conn and is a collection of sample terminal sessions that illustrate the major features of ZCPR3. I found this document useful. You can try the examples with your working ZCPR3 system. This is a good way to learn its features. This section is 41 pages. The complete "ZCPR3: The Manual" is in the works. It promises to be "lavish, typeset and over 300 pages."

Installing ZCPR3

Steps outlined in ZCPR3 "Sampler" contain too much detail concerning the multitude of options available. A user will likely be interested in this detail only after ZCPR3 is installed and some working experience is gained. A much simpler step-by-step procedure that produces a "standard" configuration could be used. A memory layout is proposed as well as all of the options have default values in the .LIB files. A generic modification to an existing BIOS could be developed that would not require access to the source code of the existing BIOS. Echelon offers an auto-install version of ZCPR3 called Z3-Dot-Com, as well as full CP/M 2.2 replacements called Z-Com and Z-System.

After some study, I decided to accept the memory layout for the maximum installation that is supplied in file Z3BASE1.LIB. I renamed this file to Z3BASE.LIB. The Installation Manual section spends 7 pages describing this file's contents. Next, the default options in Z3HDR1.LIB are fine. I renamed this file to Z3HDR.LIB. The Installation Manual section spends 14 pages describing this file's contents. So here are 21 pages you look at before you realize that it can be skipped for later appreciation.

Once the Z3BASE.LIB and Z3HDR.LIB files are defined, you are ready to assemble the ZCPR3.ASM file. This is as simple as entering one MAC command. The problem is that this command is described in section 4.5.9 while the procedure to overlay the old BIOS and CCP is mentioned in section 4.3. That is two subsections before the ZCPR3.ASM file is assembled. This lack of good organization in the Installation Manual as well as those infernal X.X.X section numbers just serves to add unnecessary complication to the installation process.

With ZCPR3.HEX created, you are now ready to work on the BIOS modifications. The reason that this is required is that ZCPR3 expects to find initializing information in the "system segments" areas of memory when it first comes up. The only easy way to do this is to initialize these areas in the cold boot BIOS code. As I mentioned, excerpts from a modified BIOS are shown in the installation manual. These excerpts cover 15 pages (the code is heavily commented).

I chose to proceed as if I did not have access to my BIOS source code. The memory layout specified in Z3BASE.LIB provide 3.5k-bytes for the BIOS. Since the BIOS I use takes less than 3k-bytes, I decide to use the 0.5k-bytes at the end of the BIOS as patch space. I typed in the cold boot code presented in the excerpts and assembled it to run at the patch address. In the patch procedure to overlay the CCP and BIOS, I read the cold boot patches into the proper proper and then patched the cold boot at the start of the BIOS to use the new cold boot code. At the end of the cold boot code, I patched a jump to the original cold boot code. This can be illustrated in Figure 1.

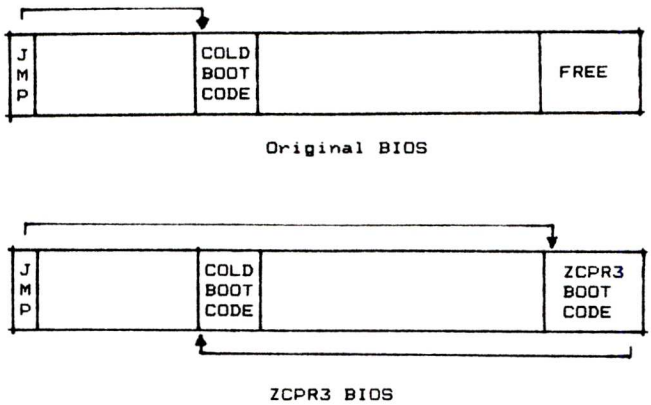


Figure 1

I think this method is sufficient to get started. The only feature not supported is character I/O redirection. Since I have not implemented an IOP (I/O Processor segment), I didn't mind. The BIOS modifications required to implement I/O redirection are not mentioned in the text; they have to be done. gleaned from the listing excerpt, The details of writing an IOP are not included in the installation manual (they shouldn't be included). I suspect the full manual will elaborate.

By this point, with the ZCPR3.HEX and CBIOSZ.HEX files available, you are ready to patch the system image. The system's size may first have to be adjusted to match the memory data in Z3BASE.LIB. The patching steps require that the system image be obtained with SYSGEN, saved to a file, this file loaded with DDT or SID, the two HEX files above patch in with the proper offsets (i.e., you have to know where your SYSGEN puts the CCP, BDOS, and BIOS in the system image), the patched image saved to a file and finally SYSGEN run to get the patched image on the system tracks of the disk. This is the kind of stuff books are written about. Anybody who has done this remembers it well, the others probably don't want to attempt it. But it's not quite as bad as it sounds. The ZCPR3 installation manual takes a good crack at an explanation by way of an example "patching" session. It can be done. Gather your system's facts and study the example — and maybe buy an "All you ever wanted to know about CP/M" type book.

Now you can boot up ZCPR3. When I did I found that I didn't have a DIR command. (What's the first thing to try after booting a new system?) The Z3HDR.LIB file had specified that the DIR command not be resident. Rather an RCP (Resident Command Processor) segment could provide DIR or one of the may directory transients (utilities) supplied could be used.

I proceeded to assemble the system segments for the environment, named directories, resident command processor and flow control processor. Each of these assemblies uses an .ASM and corresponding .LIB file. The .LIB files contains more options that the installation manual details in 22 pages. As before, I chose to go with the examples or defaults supplied. I skipped the IOP segment since this needs to be customized to conform to your particular system's I/O. Finally, I selected my terminal from a list supplied by running the TCSELECT command. This provides the information needed for the screen-oriented utilities. These system segments (including the terminal description) are loaded into their proper locations above the BIOS with the LDR command. This has to be done once after each cold boot; but the LDR and other commands can be put into one alias file and automatically executed on cold boot.

The final installation step is to run the Z3INS program that installs the system's environment description into each utility program's code. This is simple since a file containing a list of the installable utilities is provided as one argument to the Z3INS command. The other argument is the system's environment file SYS.ENV that was created by the assembly of the environment segment program.

By putting all of the system segment files and utilities on one disk, you can generate a ZCPR3 system disk. Now all of the features of ZCPR3 are available.

Major features

1. DIRECTORIES: In effect, the CP/M 2.2 disk/user concept is allowed to be named. This makes the disk/user idea more useful, but it's not a Unix or MS-DOS directory. It's flat, not hierarchical, i.e., you can't have a directory as the contents of another directory. (ed note: by using the CD, change directory, command which auto-loads ST, start alias, a new named directory is loaded permitting hierarchical, structured disk directories. New directories are created using MKDIR utility command.) I have never used the DU fea-

ture, who can remember what is in user 5 versus user 13. But if disk A: user 5 is called HELP: and disk A: user 13: is called ROOT:, I can remember what's there and I will use the DU feature by using named directories. This is great. CP/M Plus could use the named directory idea since it suffers from the same unfriendly user numbers. Another good feature of named directories is their use can be password protected.

2. COMMAND LINES: Multiple commands per line are provided. This same Unix shell delimiter ';' is used. CP/M Plus has this, but with a different delimiter. One missed feature of CP/M Plus is command line editing. Once you hit the RETURN key and send a command line off to ZCPR3, if an error occurs due to a typo, you will have to retype the whole line (ed note: using one of the ZCPR3 error handlers, any single command can be re-entered without changing any other in the line).

3. COMMAND PROCESSING — THE PATH: Command search paths can be specified. This is the old (original) ZCPR feature made better. The search path can be modified at any time.

A flow control package can be used if one is installed. The FCP can be changed at any time. This means logic can be put into command files that can control the flow of command execution. This doesn't work very smoothly in CP/M 2.2 since commands (except ZCPR3 utilities) don't return status codes. CP/M Plus has rudimentary status return checking. So in ZCPR3, the command files have lines like:

```
IF INPUT Type N or F to ABORT if Errors Exist
```

that solicit terminal input.

Resident Command Packages can be added to give fast built in commands like **CP**. You can have several RCPs and change them at any time. The **H** command will always list the available RCP commands.

The command processing search always ends with either an error handler or "extended" command processor. The error handler may default to simple **?** or a sophisticated screen-oriented menu error handler (several are supplied) may be used. I did not try the extended command processor; but it seems to be an opportunity to have a command processor like ZEX take over if all else fails. One penalty for this extensive search path is speed. The disk can grind while all of the possibilities are checked.

4. ALIASES: This is a clever idea that provides easy to develop command(s) packages, Parameter passing is supported as with SUBMIT. This is like a macro preprocessor for command lines. The ALIAS command solicits command lines and produces a .COM type file, hence the resulting file doesn't need any command interpreter to run. aliases is useful for defining the STARTUP (at cold boot) command.

5. SHELLS: ZCPR3 shells are front-ends that provide a different user interface in place of the normal ZCPR3 prompt. Several "menu" shells are provided that can be used to display a user written set of menus, accept input and execute the appropriate commands. A general purpose SH shell is provided that supports variable support.

6. Z3TCAP: This is the ZCPR3 TCAP (Terminal Capability) utility that allows ZCPR3 to have easily modified screen-oriented utilities. The TCSELECT program allows a selection to be made of any of many (ed note: presently 56) predefined terminals and packaged computers. TCMAKE allows terminals and computers not already supported to be added.

7. SECURE SYSTEMS: ZCPR3 offers password protection for selected command and named directories. With the DU form disabled, a secure environment can be defined. The directories that a user can access are the ones whose name is know, and some of these may be password protected.

Conclusion

This covers the major features of ZCPR3. As far as some limitations that I see, the reduced size of the TPA required to supply ZCPR3 with its buffers and such is no big deal. All of the ZCPR3 utilities require a lot of disk space. A double density 8-inch floppy (500k-byte capacity) can barely handle it. Double sided 8-inch (1 megabyte) floppy will do it. A hard disk would be a big help. An Echelon newsletter points out that hard disk storage is cheaper than RAM (the problem is that you have to buy so much of the hard disk storage, i.e. 10 megabytes or more).

For the price, the utility of ZCPR3 is a bargain. The utilities represent the best of the public domain stuff plus the screen-oriented versions of these utilities make them very easy to use. The installation problems can be made easier for the do-it-yourself types, and the auto-install versions from Echelon should allow ZCPR3 (ed note: and also the Z-System) to reach commercial and casual users.

C And The Godbout Disk 1 Controller

by Edward Heyman

In his article in the 1982 September/October Microsystems Anthony Skjellum made the point that the C programing language is a viable alternative to assembly language for utility programs. My experience is that C not only reduces the time required to develop correct programs versus assembly language, it also makes it simpler to write programs that are "user friendly". This article discusses a practical example of the use of C to write disk utility programs. It also provides some insight into the operation of the Intel 8272 controller chip and the Godbout Disk1 controller board as well as providing several useful utilities and a command language to use to create your own utilities.

Although some of the utilities could be written with calls to the CP/M BDOS or BIOS, directly accessing the controller lets you use faster and more powerful commands. For example, entire cylinder reads and writes may be done. Also there are no provisions in the CP/M BDOS or any BIOS that I have seen, to format a disk.

The Objectives

I recently upgraded my S-100 system with a Godbout Disk1 DMA controller and Qume double sided eight inch drives. Although Godbout supplies a format and a Copy program, I wanted to be able to format individual tracks, to verify a disk and locate any bad sectors and I wanted a faster copy program. I also use the UCSD 'p' system and needed some utilities for special disk formatting and system generation. However my most important objective was to understand how the controller worked.

The Strategy

The first step was to read the Godbout and Intel literature in the Disk1 manual. This literature was very helpful but, as I was to learn, did not tell the whole story. I then wrote routines to implement simple disk operations, placing the routines in a program that let me test their operation. I used a lot of printf() statements so that I could trace the progress of the program and understand what the controller was doing. Once a routine was shown to be correct I placed it in a library and proceeded to the next routine.

When the library contained all the primitive disk routines I began a second library that contains more powerful disk commands based on the disk primitives. Once the second library was complete writing the utilities was a simple task.

The Hardware

The Disk1 board uses the Intel 8272 / NEC 765 disk controller chip. The C routines that I developed are applicable, with some modifications, to other boards using this chip.

Communication between the CPU and the Disk1 is done through three IO ports. Two of these ports are for the 8272 chip and the third is implemented on the Disk1 board. The ports are as follows:

DISK1 CONTROL PORT FUNCTION

PORT NAME	PORT FUNCTION READ	WRITE
FD_STATUS	CHIP STATUS REGISTER	
FD_DATA	RESULTS DATA	COMMAND DATA
DISK1_PORT	BOARD INTERRUPT STATUS	DMA ADDRESS

Each command to the disk controller has a command phase, an execution phase and a result phase. During the command phase, a series of bytes is sent to the controller data port. When the controller receives the last byte of the command, it enters the execution phase. When execution is complete the interrupt status is set and the result data bytes must be read by the cpu from the fd_data port. The major commands that the 8272 chip understands are:

Command	Number of command bytes	result bytes
Specify	3	0
Sense Interrupt Status	1	2
Sense Drive Status	2	1
Recalibrate	2	0
Seek	3	0
Read ID	2	7
Format Track	6	7
Read Data	9	7
Read Track	9	7
Write Data	9	7

With single-sided floppies the term "track" is defined. With two sided disks the usage is less clear. I will use the term cylinder to mean the position of the heads with respect to the disk and the term "track" to mean the location that a single head can access when located at a specific cylinder. In general we can find the cylinder and head for a two sided disk knowing the track by:

```
cylinder = track / 2;   /* integer divide                */
head     = track % 2;   /* remainder of integer divide   */
```

For single-sided disks the head is always 0 and the cylinder is equal to the track.

Getting Started

The first routines that I wrote were three routines to interrogate the status ports and wait until they are clear. These routines are cmdstat() which signals that the 8272 is ready to accept a command byte, intstat() which signals the completion of the execution phase, and resultstat() which signals that the 8272 is ready to have a result byte read. The code for cmdstat() is given in listing 1.

Next I wrote routines that would interpret the results returned by the 8272 after command execution. The 8272 has four status registers named Status Register 0 to Status Register 3. Each of these registers is one byte wide. Almost every bit in each status register byte has a meaning. Not every command returns results in all registers. In addition to the status registers, with some commands, the 8272 returns information about the current track, current sector, current head and the number of bytes per sector as, single byte values. The routines st0(),st1(),st2(), and st3() display the mnemonic for the individual bits along with their values for the four status registers.

The code to display the results re-

LISTING 1

```
/* reads the 8272 status port untill the chip is ready for a command */
/* byte, must be done before each command byte is sent to the 8272   */

int     cmdstat()
{
    char        c;

    while ( (c = inp(fdcstat)) < 0x80);
    return(TRUE);
}
```

LISTING 2

```
/*      print value of 8272 status register 0's bits                  */

st0(byte)
char    byte;
{
char    drive,hd,nr,ec,se,ic;

    drive = byte & 3;                   /* drive number              */
    hd    = (byte >> 2) & 1;            /* head address ie; 0 or 1   */
    nr    = (byte >> 3) & 1;            /* not ready if 1            */
    ec    = (byte >> 4) & 1;            /* equipment check           */
    se    = (byte >> 5) & 1;            /* seek end                  */
    ic    = (byte >> 6) & 3;            /* interrupt code, does not  */
        /* work as per 8272 data sheet for read and write commands   */
        /* since Godbout did not implement the terminal count line   */
    printf("\n ST0>  %s %u %s %u %s %u %s %u %s %u %s %u \n",
        "drive = ",drive,
        " head = ",hd," nr = ",nr," ec = ",ec," se = ",se," ic = ",ic);
}/* st0 */
```

LISTING 3

```
/*              prompt for and return drive number                    */
/*   drive may be entered as 'a','b','A','B','0' or '1'               */

char    getdrive()
{
    char        drive;

    printf("\nEnter drive designation ");
    do
    {
      drive = getchar();
      if( (drive == 'A') || (drive == 'B') )
        drive -= 'A';
      else if( (drive == 'a') || (drive == 'b') )
             drive -= 'a';
           else if( (drive == '0') || (drive == '1') )
                  drive -= '0';
                else
                {
                  putchar('\b');
                  putchar(' ');
                  putchar('\b');
                }
    }
    while(drive != 0 && drive != 1);
    return(drive);
} /*getdrive*/
```

LISTING 4

```
/*                              RECAL.C                               */
/*      program for testing routines recal() and senseintstat()       */

#include bdscio.h
#include cdisk.h

main()
{
    char        drive;
    int         temp;

    while(TRUE) {
        drive = getdrive();
        printf("\n drive = %u  \n",drive);
        recal(drive);
        }/*while*/
}/*main*/

senseintstat(bytes)
char bytes[];
```

turned in status register 0 is given in listing 2. Bit fields are not implemented in BDS 'C'. I therefore used the 'shift' operator to move the bits into the rightmost position and then the 'and' operator to mask out the unwanted bits. If bit fields were available a more elegant solution would be to create a structure that would permit direct access of the bits.

The five 8272 commands, listed above, that return seven result bytes all have the same result format so I was able to write a single routine, called printresult(), that displays all of the data returned by these 8272 commands.

To simplify writing programs to test the procedures, I wrote four routines to allow the drive, cylinder, sector and head to be input from the console. These routines are getdrive(), getcyl(), getsector() and gethead(). The code for getdrive() is shown in listing 3.

Doing Something

Now I was ready to write routines that did something. The first routine was recal(). This routine places the head(s) of the selected drive at cylinder zero. The 8272 command for recalibrate requires two command bytes and does not return any result bytes. To determine if the recalibration was successful it is necessary to use the 8272 Sense Interrupt Status command. This command returns Status Register 0 and the current cylinder number. The last two bits of Status Register 0 are called the interrupt code (IC). The values of IC have the following meanings:

0 = Command completed successfully.
1 = Execution started but not successfully completed.
2 = Invalid command issued, execution never started.
3 = Command aborted because ready signal changed.

I used the value of IC and the value of the cylinder location byte to determine if the recalibration was successful. Recal() returns a true (1) if the recalibration was good. If it was not then the routine trys two more times. If the retrys are also unsuccessful then Status Register 0 is decoded for the cause of the failure and an error message is displayed.

The program that I wrote to develop the recal() routine is given in listing 4. The program has a short main section and two subroutines, recal() and senseintstat(). Once I had the program running properly I removed the debugging displays and placed the two subroutines in the library.

The next command to tackle was

dseek() . Dseek() positions the head over the specified cylinder. I called it dseek() to differentiate it from the BDS 'C' built-in seek(). Dseek() is similar to recal() except that the command phase requires a third byte, the cylinder number, to be sent to the 8272. I used the same technique, of writing a program to check the operation of the dseek() routine, this time since I had already developed the senseintstat() routine and had it in the library, I only had to link it to the test program.

Doing More

Every command to the 8272 starts with a command byte that has a specific structure. Bits 0 to 4 of this byte contain the code that tell the 8272 which command is being called. The command codes are given as #define statements in CDISK.H (listing 5). Bit six, called the mfm bit, tells the 8272 whether a single or double density operation is requested. Bit seven, called the mt bit, tells the 8272 whether the operation is to be single or double sided. Not every command uses the mt and mfm bits.

In addition to the command codes, CDISK.H also contains the global variables that I created to reduce the amount of data that had to be passed between procedures. One particular variable requires some additional discussion. The variable bps describes the format of the disk. The bytes per sector are related to bps by the formula: bytes per sector = 128 * 2^bps.

The readid() routine, which returns the density and format of the selected disk, was the next to be implemented. The 8272 read ID command requires a two byte command phase. Bit six of the first byte must be the mfm bit. But we don't know if we are doing a single or double density operation. In fact the reason to use the 8272 read ID command is to determine the density and the format of the disk. If the mfm bit is set incorrectly the 8272 read ID command will return a non zero interrupt code (IC), in status register 0, during the result phase. The way I constructed the readid() routine was to set the global variable mfm[drive] to 1 (double density), call the 8272 read ID command, then check for IC = 0 indicating a correct operation. If the operation is correct the routine sets the global variable n[drive] equal to result byte six and returns true. If IC is non zero then mfm is set to 0 (single density) and the 8272 readID command is called again. If this time IC is zero readid sets n[drive] and returns a true value. If IC is not zero then the other status registers are decoded, an error message is displayed and the routine returns false. In any event the seven byte results array is

```
/*              reads and returns the 8272 status register 0          */
/*              and the current cylinder number                       */
{
    cmdstat();                  /* wait till ready for command byte  */
    outp(fdcdata,c_rsts);       /* send command byte                 */
    resultstat();               /* wait till ready for result byte   */
    bytes[0] = inp(fdcdata);    /* get status register 0             */
    st0(bytes[0]);              /* print Status Register 0           */
    resultstat();               /* wait till ready for result byte   */
    bytes[1] = inp(fdcdata);    /* get cylinder number               */
    printf("\ncylinder = %u\n",bytes[1]);
}/* senseint stat */


int     recal(drive)
char    drive;
/*      move the head to cylinder zero, return TRUE if successful    */
/*      print error message if unsuccessful                           */
{
int     k;                      /* try counter                       */
char    bytes[8];               /* array to hold results             */

for(k=0; k < 3; K++)
{
    cmdstat();                  /* wait till ready for command byte  */
    outp(fdcdata,c_reca);       /* send command byte                 */
    cmdstat();                  /* wait till ready for command byte  */
    outp(fdcdata,drive);        /* end of 8272 recalibrate command phase*/
    intstat();                  /* wait till execution phase complete*/
    senseintstat(bytes);        /* check if recal ok                 */
    /*  check for satisfactory completion and at cylinder 0     */
    if( ((bytes[0] & no_err) == 0x00 ) && (bytes[1] == 0)) return(TRUE);
}/* or */
printf("\nRecal error drive %c ",drive+'A');
if ( (bytes[0] & ds_err) != drive ) printf("incorrect drive select\n");
if ( (bytes[0] & nr_err) != 0 ) printf("not ready\n");
if ( (bytes[0] & eq_err) != 0 ) printf("equipment error\n");
return(FALSE);
}/* recal */
```

LISTING 5

```
/*                              CDISK.H                               */

#define TRUE    1
#define FALSE   0

/* the following sets the conditional compile in setdma() so that the */
/* tpa(transient program area is in page 0 for CPM 2.2 and page 1 for */
/* CPM 3. If you are using with CPM 2.2 or a non-banked CPM 3 set to  */
/* FALSE                                                              */
#define CPM3    TRUE

/*                          DISK1 PORTS                               */
#define fdport  0xc0            /* base address of disk controller   */
#define fdcstat fdport          /* 8272 status port                  */
#define fdcdata fdport + 1      /* 8272 command and results data port */
#define fdma    fdport + 2      /* Disk1 DMA port (write)            */
#define ints    fdport + 2      /* Disk1 interrupt status port (read) */

/*                        8272 COMMAND CODES                          */
#define c_rtk   0x02            /* read a track                      */
#define c_spec  0x03            /* specify                           */
#define c_dsts  0x04            /* sense drive status                */
#define c_wrat  0x05            /* write data                        */
#define c_rdat  0x06            /* read data                         */
#define c_reca  0x07            /* recalibrate                       */
#define c_rsts  0x08            /* sense interrupt status            */
#define c_rdid  0x0A            /* read ID                           */
#define c_form  0x0D            /* format                            */
#define c_seek  0x0F            /* seek                              */

                        8272 ERROR MASKS
#define ds_err  0x03            /* incorrect disk select             */
#define nr_err  0x08            /* not ready error                   */
#define eq_err  0x10            /* equipment error                   */
#define no_err  0xc0            /* no error                          */

/*                        GLOBAL VARIABLES                            */
/*      globals have the form variable[drive] where drive is 0..3     */

char    mt[4];                  /* two side operation = 1, one side = 0 */
char    mfm[4];                 /* double density = 1, single density = 0 */
char    sk[4];                  /* skip (not used) always 0          */
char    bps[4];                 /* bytes per sector 0,1,2,3 for Godbout format */
char    eot[4];                 /* final sector number of track      */
char    gpl[4];                 /* gap length                        */
char    dtl[4];                 /* data length                       */

int     x,y;                    /* cursor coordinates used by iolib.c */
```

LISTING 6

```
/* Getresult() routine reads the results from a read,write,format or */
/* readid command returns TRUE for a good read or write              */

int     getresult(bytes)
char    bytes[];
{
  int   k;                          /* byte counter                  */

  for (k = 0 ; k < 7 ; ) {
    resultstat();                   /* wait till ready for result byte */
    bytes[k++] = inp(fdcdata);      /* read result byte              */
    }
  return((bytes[1] == 0x80));       /* return TRUE if end of cylinder */
}/* getresult */
```

LISTING 7

```
/*              DENSITY.C                                            */
/* program to return number of sides, density and format of a disk   */

#include bdscio.h
#include cdisk.h

main(argc,argv)
char    **argv;
{
  char  cyl,hds,drive;
  char  bytes[8];
  int     i;

  printf("\ndensity version 1.3\n");
  drive = getdrive();                   /* request drive            */
  hds= 0;                               /* select head 0            */
  cyl = 2;                              /* select cylinder 2        */
  dseek(drive,cyl);                     /* position head            */
  getmt(drive);                         /* find number of sides     */
  readid(drive,hds,bytes);              /* find density and format  */
  printf("\ndrive %c is ",(drive+'A'));
    if (mt[drive]) printf("Double Sided, ");
    else printf("Single sided, ");
    switch (n[drive]) {
        case 0  : printf("Single density with 128 Byte sectors\n");
                  break;
        case 1  : printf("Double density with 256 Byte sectors\n");
                  break;
        case 2  : printf("Double density with 512 Byte sectors\n");
                  break;
        case 3  : printf("Double density with 1024 Byte sectors\n");
        } /* switch */
} /* density */
```

LISTING 8

```
/*              segment of setparam() code                           */

    else                /* double density */
        switch (n[drive]) {
            case 1      : gpl[drive] = 0x0e; /* gap length          */
                          eot[drive] = 0x1a; /* last sector on track */
                          dtl[drive] = 0xff; /* no meaning in dd     */
                  break;
            case 2      : gpl[drive] = 0x1b; /* gap length          */
                          eot[drive] = 0x0f; /* last sector on track */
                          dtl[drive] = 0xff; /* no meaning in dd     */
                  break;
            case 3      : gpl[drive] = 0x35; /* gap length          */
                          eot[drive] = 0x08; /* last sector on track */
                          dtl[drive] = 0xff; /* no meaning in dd     */
                  break;
            } /* case */
}/* setparam */
```

returned to the caller. The readid() command is the first to be discussed that returns seven result bytes. To read those bytes from the 8272 I created the routine getresult() which is shown in listing 6. I'll say more about getresult() later.

Using readid() we have the density and the number of bytes per sector,but we still need to know if the disk is single- or double-sided. The number of sides on a disk can be determined by looking at the two-sided bit in Status Register 3. The 8272 Sense Drive Status command returns Status Register 3. Routine getmt() issues the 8272 Sense Drive Status command, reads the two result bytes, and then sets the global mt [drive] byte in accordance with the two sided bit in Status Register 3.

At Last A Program

We now have all we need to write the program Density. This program reports the number of sides, the density and the number of bytes per sector of a disk in the selected drive. The program is given in listing 7.

Going Further

All of the read and write commands as well as the format command require that the Disk1 be told where in memory to read from or to write to. The setdma() routine does the job. As it is written, all addresses are on 64k page 0. That can easily be changed if you want to address other pages, since the Disk1 DMA register is a 3 byte register.

The 8272 read and write commands require a 9 byte command. I have already discussed the content of the first byte. The second byte tells the 8272 which drive and head to use. The next four bytes are the cylinder, the head, the sector and the number of bytes per sector. The remaining three bytes describe the disk format. The first of the three final bytes is called EOT and is the number of the last sector on a side (the first sector is 1). The next byte is GPL, the gap length, which is the spacing between sectors. The final byte is the data length, DTL. DTL has significance only when bps is 0 (ie; 128 bytes per sector).

The values of EOT, GPL, and DTL are determined by the values of mfm (density) and n (bytes per sector). Routine setparam() when called after readid() will define the values of the global variables EOT, GPL and DTL. Listing 8 is a segment of the setparam() code.

Before I could implement the read or write commands I had to create a routine to fill the command byte array with the nine command bytes. Four very similar commands do the job. The routines are called set??cmd() where ?? is:

rs : to read one or more sectors
rc : to read a cylinder
ws : to write one or more sectors
wc : to write a cylinder

Listing 9 contains the code for setrscmd(rdcmd,drive,cyl,sector, sectors,hds). Rdcmd is an 9 byte array to hold the read command.

Using the rs or ws routine permits any number of sectors on a side to be read or written. The first sector to be read or written is passed to the routine as sector and the number of sectors to process is passed as sectors. Note that when I talk about sectors I mean physical sectors which can be 128, 256, 512 or 1024 bytes long, not CPM

LISTING 9

```
/*              Setrscmd() fills the 9 byte read command array    */

setrscmd(rdcmd,drive,cyl,sector,sectors,hds)
char    rdcmd[],drive,cyl,sector,sectors,hds;
{
    rdcmd[0] = (mfm[drive]<<6) | (sk[drive]<< 5) | c_rdat;
    rdcmd[1] = (hds << 2) | drive;      /* head and drive data  */
    rdcmd[2] = cyl;                     /* cylinder number      */
    rdcmd[3] = hds;                     /* head                 */
    rdcmd[4] = sector;                  /* first sector to read */
    rdcmd[5] = n[drive];                /* disk format          */
    rdcmd[6] = sector + sectors - 1;    /* last sector to read  */
    rdcmd[7] = gpl[drive];              /* gap length           */
    rdcmd[8] = dtl[drive];
}/* setrscmd */
```

LISTING 10

```
/* perform a read operation either sector or cylinder. Must have set up*/
/* the rdcmd[] array before calling this function. Returns TRUE if read*/
/* was successful FALSE if an error occured.                           */

int     dread(rdcmd,bytes)
char    rdcmd[];                /* array contaning the read command */
char    bytes[];                /* array to recieve the results     */
{
    int j;                      /* byte counter                     */

    for (j = 0; j < 9;j++)      /* send 9 bytes                     */
        {
        cmdstat();              /* wait till ready for command byte */
        outp(fdcdata,rdcmd[j]); /* send read command byte           */
        }
    intstat();                  /* wait till end of execution phase */
    if(getresult(bytes)) return(TRUE);
    else {
        printf("\nread error drive %c head %u ",
            ( (bytes[0]&3)+'A' ),( (bytes[0]>>2) & 1) );
        geterror(bytes);
        return(FALSE);
        }
}/* dread */
```

logical sectors that are always 128 bytes long.

The 8272 chip has the capability of doing multitrack reads or writes. That is, it can start on side 0 and complete the operation on side 1. The read cylinder (rc) and write cylinder (wc) commands require that bit seven of the first command byte be set to mt [drive] to enable (value = 1) or disable (value = 0) a multitrack operation.

The routine dread() (listing 10) sends the command bytes to the 8272, waits until the execution phase is complete, and then reads in the result bytes from the 8272. The dwrite() routine operates in the same way as dread().

Now putting it all together, the sequence for a cylinder read is:

```
setdma(buffer);                     /* mem location to start data   */
dseek(drive,cylinder);              /* position head                */
getmt(drive);                       /* determine number of sides    */
readid(drive,hds,bytes);            /* determine n and mfm          */
setparam(drive);                    /* determine EOT,GPL and DTL    */
setrccmd(rdcmd,drive,cylinder);     /* fill command array           */
dread(rdcmd,bytes);                 /* execute command              */
```

I put the last six commands together to make a single readcyl() routine that does the whole job after setdma() is called. A similar routine was developed to write a cylinder and named writecyl().

If only a track is to be read, the

LISTING 11

```
/* Geterror() prints  an error message for an incorrect read or write*/
/* command, detailing the error. Must call getresults(bytes) first.  */

int     geterror(bytes)
char    bytes[];                /* array containg the results bytes  */
{
if (( bytes[0] & 8) != 0) printf("not ready \n");
if (( bytes[1] & 0x7f) != 0) {
    if ((( bytes[1] & 1) != 0) && (( bytes[2] & 1) == 0))
                        printf("missing ID address mark\n");
    if (( bytes[1] & 2 ) != 0 ) printf("write protected\n");
    if (( bytes[1] & 4) != 0) printf("no data transfered \n");
    if ((( bytes[1] & 0x20) != 0) && ((bytes[2] & 0x20) == 0))
                printf("crc error in ID field\n");
    }
if (( bytes[2] != 0)) {
    if (( bytes[2] & 1) != 0) printf("missing data address mark\n");
    if (( bytes[2] & 0x20) != 0) printf("crc error in  data field\n");
    if (( bytes[2] & 0x10) != 0) printf("wrong cylinder\n");
    }
return(TRUE);
}/* geterror */
```

LISTING 12

```
/* Segment of setform() command that sets up format command sequence */
/* to be read by the controller as it formats each sector.           */

for(sector=1,k = 0; sector<=eot[drive]; sector++){/* do for each sector*/
        buffer[k++] = cyl;                          /* cylinder   */
        buffer[k++] = hds;                          /* head (side)*/
        buffer[k++] = sector;                       /* sector     */
        buffer[k++] = n[drive];                     /* format     */
    }/* for */
}/* setformat */
```

command sequence of readcyl() is modified by replacing setrccmd() with setrscmd(rdcmd, drive, cylinder,head,1,eot[drive]). Here I have set the first sector to read as 1 and the number of sectors to read as eot[drive], which will read all the sectors on one side. The resulting routine is readtrack(). Its companion is writetrack().

How About Formatting?

The only primitives left to do are those for the formatting operation. The routines required to format a track are similar to those for the read or write operations. The setformat() routine is similar to the setparam() routine. A separate routine is required because the value for GPL is different for formatting and read/writing . It is also necessary to supply the cylinder,head, sector and bytes per sector (bps) for each sector in the track to be formatted. During the formatting operation the 8272 reads each set of four values prior to formatting the sector. The code of setformat() is shown in listing 12.

The setformcmd() is similar to the setrtcmd() except that the command is only six bytes long. The last byte in the command array is the value of the data mark (E5). Routine form() is similar to dread() and dwrite().

Formtrack() puts together all the routines required to format a track. Finally the routine formcyl() formats an entire cylinder. Multitrack operations are not allowed with the 8272 format command. A cylinder of a two sided disk therefore requires two calls to the form() routine, once with hds set to 0 and once with hds set to 1.

On To Higher Levels

I have now described all of the primitive routines. We could at this point write the utility programs; however, several operations of the same form appear in the utilities and it is worthwhile creating some higher level routines. The higher level routines were put in library 2.

The first of the high level routines is verifycyl(). This routine sets the DMA address then does a cylinder read. If the read is successful, verifycyl() returns true, else it returns false. If verifycyl() returns true, we know that the cylinder is properly formatted and can be read or written to.

If we find an error on a cylinder we would like to know which sectors are defective. The routine, versectors(), does that job by reading each sector, one at a time, and reporting any errors that are found. Each side of a two sided disk is handled separately.

The high level routine, Verify(), combines verifycyl() and verifysecs() to examine an entire disk and report all defective tracks and sectors. Verify() first checks all 77 cylinders and collects all defective cylinders in an array named badcyls[ ]. If any errors are found, verify() calls verifysecs() for each of the bad cylinders.

Reader() reads multiple cylinders into a buffer. It simply calls readcyl() the required number of times.

Compare() compares the contents of two buffers. The buffers are assumed to be cylinders containing sectors. If differences in the buffers are found, the cylinder and sector numbers of the differing sectors are displayed on the console. If no difference in the buffers is found, true is returned.

Dmpsec(start,chunk) prints one physical sector to the console in ddt format. The dump starts at location start and the size of the physical sector is chunk.

Dmpcyl() prints an entire cylinder to the console, one sector at a time. It first calculates the size of the physical sector, then determines the total number of sectors to print. Dmpcyl() calls dumpsec() to do the printing.

Now The Utilities

Writing a program to verify a disk is now merely a matter of command line processing and calling verify().

The program formtrk formats a single track. After processing the command line arguments, the program prompts for the cylinder number, head and the number of bytes per sector. Knowing the bytes per sector, the program sets the value of mfm[drive]. The track is then formatted. If an error is encountered in the formatting operation, the contents of the status register are printed, and the program is aborted. If the formatting goes well, the track is verified with verifysecs(). Among other things, formtrk allows you to format the second side of track 0 which the Godbout format program does not do.

Dcopy is a disk to disk copy program. It reads and writes four tracks at a time for double density disks (four cylinders for single sided drives, two cylinders for double sided drives) and eight tracks for single density disks. The program gives the user the option of verifying the copy operation. Without the verification the program copies a disk in less than half the time required by the Godbout supplied routine. I have yet to find a compare error when making a copy.

Dumpcyl reads a cylinder from a disk and displays each physical sector on the console in ddt format. Readsec reads an individual sector from the selected disk and displays it on the console.

How To Fix A Disk

The repair utility is useful when you have formatting errors on one or more sectors of a cylinder. The program first determines if the errors are on the head0 side or the head1 side or both. Next the program reads the bad track(s) into a memory buffer, sector by sector. Bad sectors are filled with data marks (e5

hex). From this point on we are vulnerable to program failure since once we reformat the track the only place that has data is the memory buffer and termination of the program will result in loss of the entire track. To insure the greatest chances of success I do a loop controlled by verifysec(). If verifysec() returns a false, another attempt (up to 3) to format the track is made. After the track(s) are reformatted three attempts to write the buffer back to the track with writetrack() are made. If these attempts fail, probably as the result of permanent disk damage, a sector by sector write is done to recover as much data as possible.

How To Recover A Directory

Having a disk sector failure is always a traumatic experience. If the sector happens to be on the directory track, the problem is particularly severe, since the disk can't even be accessed. One way to recover some of the files is to use the repair utility. In this case only the files whose directory entries are on the bad sector(s) are lost. With 128 byte sectors, only four files would be lost. With 1024 byte sectors, however, 32 files would be lost. Two sided drives offer a valuable technique to protect against these occasional problems.

The technique makes use of the fact that side 1 of cylinder 1 (track 3) is not normally used. We can use this track to store an image of the directory track. The utility savedir does that job. Frequent use of savedir will insure that the directory backup is current. When that terrible moment comes and we have a directory failure the program fixdir comes to the rescue. Fixdir works much like repair except that instead of filling bad sectors with data marks the bad sectors are replaced with their backups from track three. When the repair is completed a list of the files in the affected sector(s) is displayed on the console.

Non CP/M Applications

I also run the UCSD 'P' system. My bios requires that I use two systems tracks. For simplicity I format both tracks as single density 128 byte sectors. The program pform does the job. For a single sided disk cylinders one and two are single density for a two sided disk both sides of track 0 are single density. After formatting is complete the disk is verified.

The program getcode was created to allow reading sections of the systems tracks and placing the code in normal CP/M files. This program was very useful in copying the 'P' system SBOOT from its location on track 0 so that I could later relocate it to allow for a larger primary boot. It can of course, also be used for moving parts of other operating systems.

Putcode is the reverse of getcode. Putcode reads a standard CPM file and then writes it to selected tracks and sector. I used this program to place the 'P' System primary boot, SBOOT, and sbios where I needed them.

Pgen is a program for the 'P' system that does the same job that sysgen does for CP/M. The program reads the first two tracks (the systems tracks) of the source disk and writes then to the first two tracks of the destination disk. It checks to see that the disks have the same format since the primary boots are different for single and double sided disks.

IO Interface

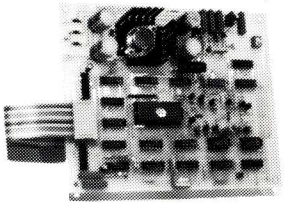
I placed all the console hardware specific routines in a file called iolib.c allowing easy modification for different consoles.

On Your Own

The complete listings are available, on disk, from the SIG/M public domain library. Using the libraries you can readily create your own utility programs and/or improve mine. If you do either I'd be interested in hearing from you.

CP/M EPROM PROGRAMMING SYSTEM

2708
2758
2716
2732
2764



2516
2732A
27128
27256
27CXX

-SOFTWARE SPECIFICALLY DESIGNED FOR CP/M COMPUTER SYSTEMS
- STAND ALONE BOARD - ELECTRONIC SWITCHING OF EPROM TYPES
- USES 24 VOLT XFMR FOR POWER - ALL SUPPLIES/TIMING ON BOARD
- DESIGNED WITH EASY TO GET PARTS - LARGE COMPREHENSIVE MANUAL

* * CENTRONICS INTERFACE * *

CONNECTS TO ANY CENTRONICS PRINTER INTERFACE - USES 8 OUTPUT DATA BITS AND ONE INPUT DATA BIT (BUSY LINE). BUSY LINE IS A HIGH SPEED SERIAL INPUT. FULL EPROM READING AND PROGRAMMING INSTALL PROGRAM PROVIDED FOR QUICK INSTALLATION TO HOST SYSTEM

* * CONTROL PROGRAM COMMANDS * *

- PROGRAM EPROM(S) FROM DISK
- READ DISK FILE INTO RAM
- READ EPROM(S) INTO RAM
- VERIFY EPROM IS ERASED
- SAVE EPROM(S) TO DISK
- PROGRAM EPROM(S) FROM RAM
- COMPARE EPROM WITH RAM
- COPY EPROM
- DISPLAY/MODIFY RAM - (MONITOR MODE) WITH 11 SUB COMMANDS FILL-DUMP-XFER-EXAMINE-MODIFY-BIAS-PROGRAM-VERIFY, ETC)

BARE P.C. BOARD WITH COMPLETE DOCUMENTATION AND SOFTWARE ON 8" SINGLE DENSITY DISKETTE $69

(ABOVE WITH COMPLETE PARTS KIT - $169)(A&T - $189)
SOFTWARE AVAILABLE FOR OSBORNE, KAYPRO AND OTHER 5 1/4 FMTS

TO ORDER SEND CHECK, MONEY ORDER, WRITE OR CALL
ANDRATECH
P.O. BOX 222
MILFORD, OHIO 45150
(513) 752-7218

CALL OR WRITE FOR MORE INFORMATION -- ADD $3.00 FOR SHIPPING
OHIO RES. ADD 5.5% TAX -- VISA/MC ACCEPTED -- $3.00 FOR COD

Still Searching For Files Without EUREKA!™ ??

You may not know it, but a disk cataloger can be a big help in managing your computer files. Why not go with the best? **EUREKA!** is a terrific time saver for ...

- Lawyers
- Software Developers
- Writers
- Teachers
- Project Managers
- Accountants
- Researchers
- Secretaries
- Consultants
- Journalists

People who try **EUREKA!** love it.....

"Just started cataloging with comments - Great Idea" GR-MI

"Great time saver in locating material on disks." WB-NY

"Your manual is the best written I have ever seen." MT-NS

"We finally chose EUREKA! ... largely because it has the ability to read comments directly from a file ... EUREKA! is easy to learn and use, has more access and report choices, finds files by many different ways, and has an attractive price."
T. Bove & C. Rhodes, USER'S GUIDE No. 11

EUREKA!, the popular CP/M® disk cataloger
Still only $50

MENDOCINO SOFTWARE COMPANY, INC.
Dept. M -3
P.O. Box 1564
Willits, CA 95490
(707) 459-9130

add $2.50 shipping
Calif. residents add 6% sales tax
VISA & MasterCard accepted

A **EUREKA!** package is designed to run on only a single system. Licenses for additional systems (for a single user) are $15 each.
EUREKA! is a trademark of Mendocino Software Company, Inc.
CP/M is a registered trademark of Digital Research, Inc.

6 TIMES FASTER!

SuperFast Software Development Tools

INCREASE YOUR PROGRAMMING EFFICIENCY
with high-performance software development products from SLR Systems.
No other tools approach the speed or flexibility of the SLR Systems line.

"Z80ASM is an extraordinary product...",
Robert Blum, Sept. 84 DDJ

"...in two words, I'd say speed & flexibility",
Edward Joyce, Nov. 84 Microcomputing

ASSEMBLERS

- RMAC/M80 macros
- Nested INCLUDES & conditionals
- 16 char. labels on externals
- Built in cross-reference
- Optional case significance
- Phase/dephase
- Math on external words and bytes
- Define symbols from console
- Generate COM, HEX, SLR-REL, or Microsoft-REL files
- Time & Date in listing
- Over 30 configure options

Z80ASM -full Zilog Z80 ..... $125
NEW! Z80ASM+ -all tables virtual $195
NEW! SLRMAC -full Intel 8080, with Z80.LIB extensions internal $125
NEW! SLRMAC+ -all tables virtual $195

Z80 CPU, CP/M compatible, 32K TPA required.

"Z80ASM...a breath of fresh air...",
Computer Language, Feb. 85

LINKERS

- Links SLR & M80 format files
- Output HEX or COM file
- Three separate address spaces
- Load map and SID/ZSID .SYM file

SLRNK+ includes:

- All tables overflow to disk
- HEX files do not fill unused space
- Intermodule cross-reference
- EIGHT separate address spaces
- Works with FORTRAN & BASIC
- Generate PRL & SPR files
- Supports manual overlays
- Full 64K output

SLRNK -fastest memory based $125
NEW! SLRNK+ -full featured virtual $195
Combo Paks available from $199. - $299.

For additional information contact SLR Systems
1-800-833-3061, in PA (412) 282-0864
1622 N. Main St., Butler, PA 16001 • Telex 559215

MasterCard VISA C.O.D., Check or Money Order Accepted

SLR Systems

16 Bit Lisp and Prolog Implementations

Part II

by William G. Wong

The conclusion of Bill Wong's in-depth review of muLisp-82, IQ Lisp, TLC-Lisp, and micro-Prolog.

Garbage Collection and Workspace

Garbage collection is always a concern with Lisp-like systems since lists are a major resource. Performance is more important on the 16-bit implementations because the number of list cells is larger. Collecting more cells takes more time, thereby reducing the time available for computation. Garbage collection time on an 8-bit system is usually one second or less, but the 16-bit implementations can have up to ten times the number of list cells. Consequently, the garbage collection time may amount to several seconds, which can definitely interrupt real-time or interactive processing. Forcing the system to perform a garbage collection at key times can make a difference in a programs response time. For example, the performance tests run for this article were performed after forcing a garbage collection, to ensure that none would occur during the test; this precaution avoided the addition of garbage collect time to the true test time. A quick examination of the system structure for each implementation will show how the workspace and garbage collect problem was addressed.

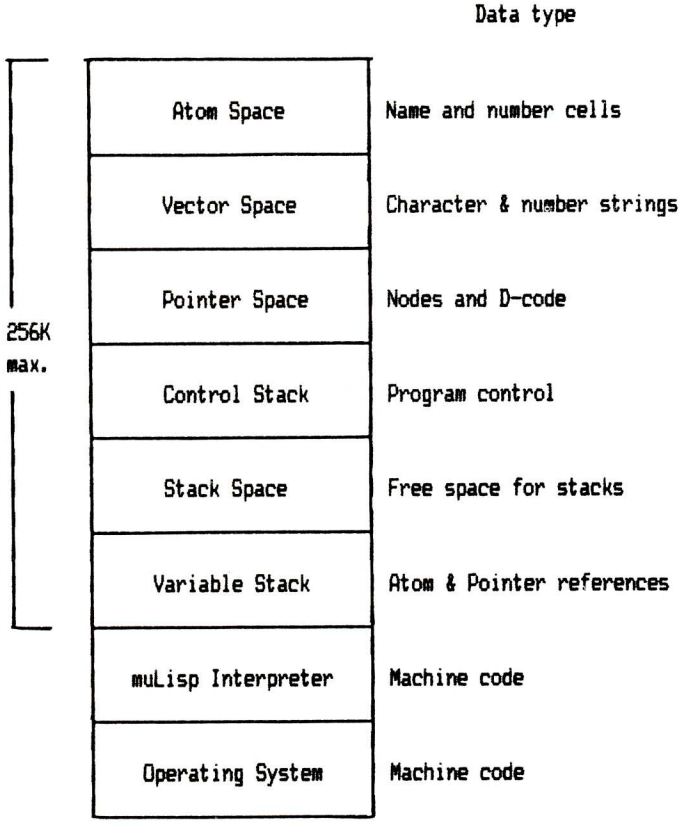
muLisp-82. The 16-bit muLisp-82 implementation is similar to the 8-bit version. It uses the same "closed pointer space" architecture which does not require type checking when accessing list cells. The workspace diagram is shown in figure 1. The workspace is divided into a number of sections containing the same type of data items. The type of data item is determined by its address as compared to the boundary address kept by the interpreter.

The biggest difference is that the list space has been increased four fold to 256 kbytes. This provides a theoretical 64K 4-byte list cells. Each cell contains two 16-bit pointers that index the data space on a cell boundary. Garbage collection time is four times slower than in the 8-bit version, and can be as much as eight seconds. In general the two pass, compacting garbage collector of muLisp-82 is faster than those of other implementations, and also leads to a faster list allocation scheme.

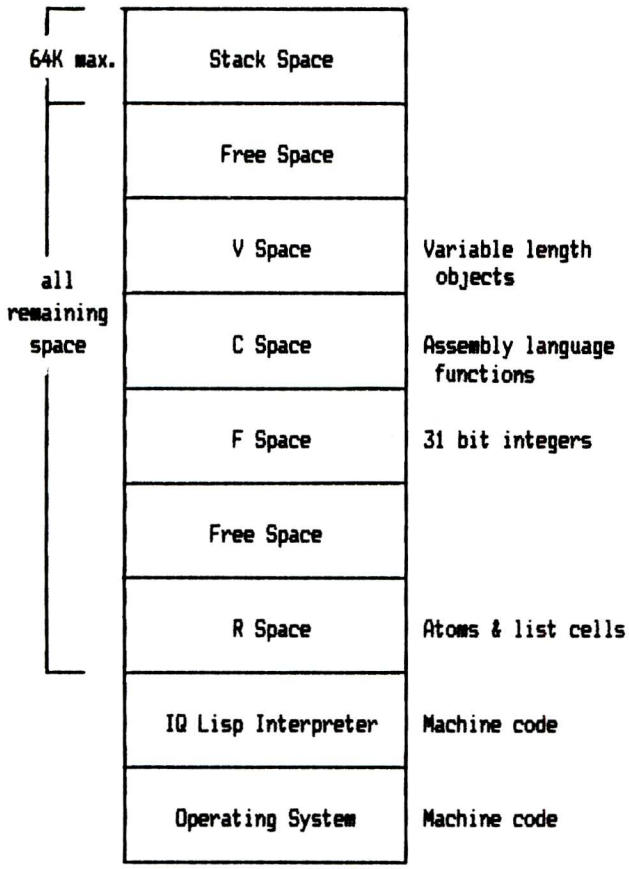
IQ Lisp. IQ Lisp also uses a two pass, compacting garbage collector and keeps similar data objects in the same partition as with muLisp. Figure 2 shows the IQ Lisp workspace layout. However, IQ Lisp uses a tagged architecture for its objects along with a 6-byte list cell. The advantage is seen in the number of different data types supported by IQ Lisp. Also, the address range of the pointers in a list cell is increased to the entire 8088/8086 address space, if you have enough memory.

In theory, IQ Lisp could support 170K list cells in a full megabyte of memory. In practice, IQ Lisp will use all the memory made available to it. Garbage collection can take more than ten seconds depending upon the amount of memory available and the amount of list space in use.

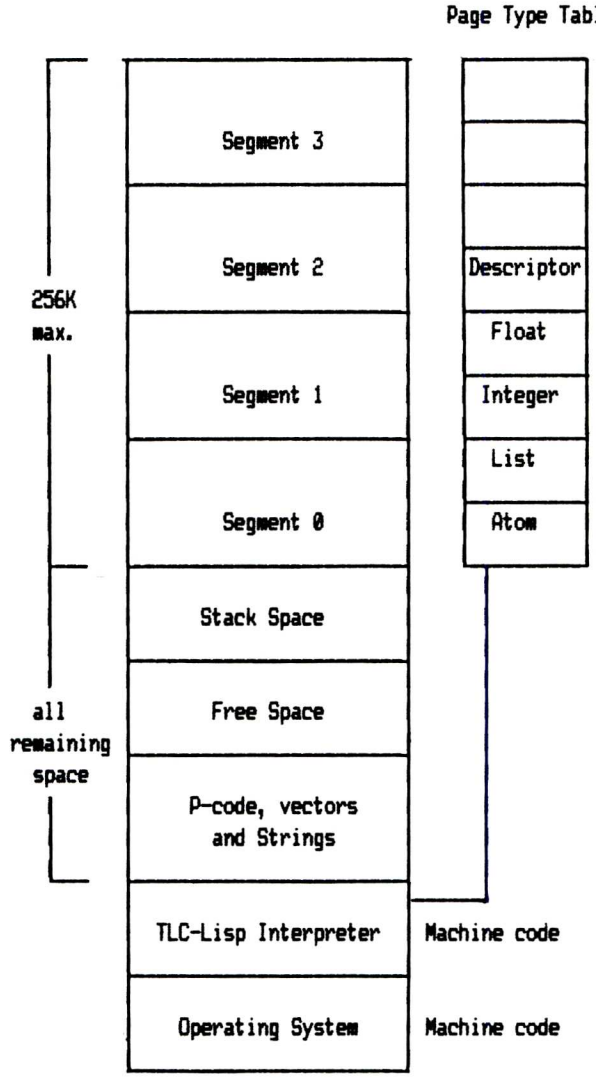
TLC-Lisp. The TLC-Lisp workspace diagram in figure 3 shows a different implementation approach. TLC-Lisp uses a combination of the muLisp-82 and IQ Lisp architectures. It is called a BIg Bag Of Pages (BIBOP) architecture. As in muLisp, the type of each object is found via



muLisp-82 Workspace
Figure 1



IQ Lisp Workspace
Figure 2



TLC-Lisp Workspace
Figure 3

its address. As in IQ Lisp, each object is tagged - but where is the tag? In fact, it is the memory page that contains an object which is tagged; this is why the Bag Of Pages is important. Each TLC-Lisp list cell consists of two 16-bit pointers that point into the 256K data space, as in muLisp-82. Each page (256 bytes/page) in this data space contains the same type of object, such as an integer or list cell, and has a corresponding entry in a data-type table that indicates the type of objects in the page. The Bag (data-type table) needs to be Big to give you enough objects of each type.

Even though it uses 16-bit pointers and a 256K list space, TLC-Lisp has an edge over muLisp-82 since objects such as strings, P-code, and vectors are stored outside of the list space. It also means that TLC-Lisp could make use of a megabyte of RAM just as IQ Lisp could, and, as in IQ Lisp, the garbage collection time can be significant with a lot of memory. Four to ten seconds is not unusual for one-half a megabyte.

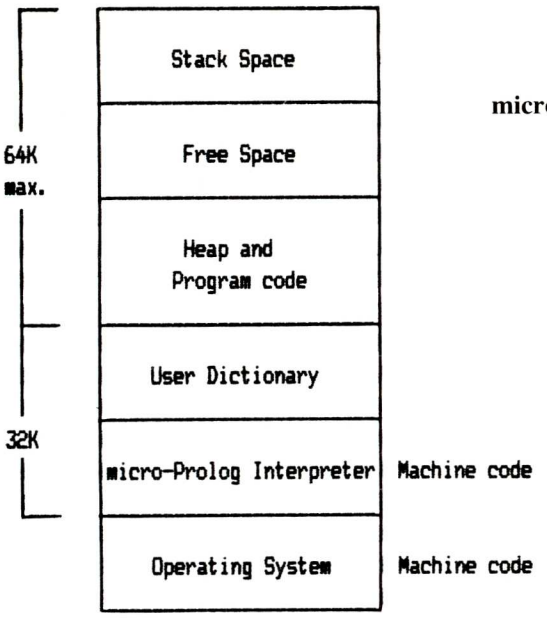
micro-Prolog. Figure 4 shows the micro-Prolog workspace structure. The small 64K data space and the 6-byte tagged list cell means a small number of list cells. The tradeoff is that garbage collection is very fast, since there is less to collect. Actually, the 8- and 16-bit implementations are very close. The only difference is that the 16-bit system has separate program (32K) and data (64K) spaces. This turns out to be almost a four-fold increase over the 8-bit version, because the operating system is also in a separate segment.

In any given system, garbage collection time will vary from collection to collection because the time is a function both of the allocation/deallocation procedures and of the current garbage state of the machine. Although a larger complement of memory takes longer to collect, garbage collection is done less often. Memory utilization will probably not be a concern for machine with 256 kbytes or less given the workspace architectures. Only IQ Lisp and TLC-Lisp take advantage of much more memory.

Assembly Language Interface

muLisp-82 does not provide any type of assembly language interface. Instead it compiles all function definitions into a distilled code called D-code. This form executes faster and is more compact than the normal Lisp function format.

IQ Lisp provides an assembly language interface that is well defined and very flexible. The documentation is quite clear, and a complete example is provided in the manual and on the distribution diskette. Assembly language programs must be created using the



micro-Prolog Workspace
Figure 4

Microsoft Macro Assembler. The loading of assembly language functions is done after Lisp is running.

TLC-Lisp does not include a machine assembly language interface; instead, a P-code virtual machine is included, along with a Lisp-based P-code assembler. A compiler is available separately. Converting functions to P-code can reduce the size of a function by as much as 50% and increases execution speed by as much as 300%.

Although the 8-bit version of micro-Prolog had an assembly language interface, the 16-bit version does not.

Performance

Performance benchmarks are always difficult to set up and more difficult to interpret. They tend to be biased even though this may be unintentional. Lisp and Prolog are particularly susceptible to this problem because the method of system implementation can make a large difference in the results. Minor changes in the test data or environment can change test times dramatically. Please examine the following results with this in mind.

The basic test used in table 1 is the empty loop. This was used as the basis for all the other tests. It was executed 5000 times, as were the other tests, to get times long enough to time accurately. The number of iterations was reduced when the elapsed time exceeded a certain point since garbage collect times came into play. The result recorded is the computed time for 5000 iterations. This may or may not be fair — but remember what I said before about benchmarks. A garbage collection was initiated before timing each test. The tests are arbitrary and do not completely test the implementations, so do not select a system on these test results alone.

Not all implementations had built-in functions for the tests performed. For example, all the Lisp implementations had a loop function, but one had to be written for micro-Prolog. Also, DR LOGO (a Lisp-like language) was included in the test set because of its turtle graphic support, which was compared to the TLC-Lisp turtle graphics.

As it turns out, the empty loop test gives a good rating for the implementations in general. muLisp-82 is the fastest, followed by IQ Lisp and TLC-Lisp, which were fairly evenly matched. micro-Prolog and DR LOGO are the slowest of those tested. The CONS test was used to see how fast list cells can be allocated. Garbage collection was not invoked during the test, so these times are not included.

Some of the stranger results come in the software floating point and list indexing tests. IQ Lisp looks much slower, but this is due to the way the functions are implemented. Instead of using machine code functions, IQ Lisp uses a combination of Lisp functions and machine code to implement software floating point support. Likewise, the IQ Lisp list index function was written in Lisp, since a built-in function was not available. Actually, IQ Lisp is very fast for most operations.

The turtle graphics turns out to be limited by the hardware — the times for TLC-Lisp and DR LOGO are quite close. This seems reasonable since the basic graphic functions are built-in machine language code. The only difference was seen on the screen. Repeatedly drawing a square caused DR LOGO to move the starting position of the square one pixel to the right and up about every 75 iterations while TLC-Lisp remained at the same position after 5000 iterations.

The slower times of micro-Prolog should also be treated with reservation, since none of the tests really exploit its features. Any pattern matching test reveals micro-Prolog as the winner even over muLisp-82.

Also, I/O tests, except graphics, were not performed since they tended to be limited by the hardware rather than to the software. Output to the screen and the printer was about the same for all systems. Disk access tests were limited by the differences in the I/O functions available under each system.

The test results do show that all the systems fair well when compared to existing interpreters such as BASIC or PASCAL. The screen editors also support the premise that the Lisp and Prolog implementations are fast and powerful enough to run general applications as well as AI applications.

Summary

The new 16-bit Lisp and Prolog implementations are more powerful and more flexible than their 8-bit counterparts. Each of the implementations reviewed provides different features and advantages.

muLisp-82 offers speed and a screen-oriented text editor. It runs under MS-DOS, CP/M-86 and variants of both. Screen manipulation is customizable, assuming terminal attributes are controlled by escape sequences. There is also an 8-bit CP/M implementation which is downward compatible. The infinite precision integers provide a powerful tool for algebraic manipulation programs, which was the original purpose for the product. On the other hand, muLisp-82 lacks a number of the more useful features found in the other implementations, such as floating point support, vectors, MACROs, and error handling. Also,

Test Description	TLC Lisp	IQ Lisp	muLisp	micro-Prolog	DR Logo
Empty loop	30	29	10	70	90
CONS test	40	40	11	150	550
Integer addition	40	37	12	105	550
Float addition	45	400	--	100	575
Integer multiplication	39	280	14	150	580
Float multiplication	43	380	--	150	575
Assignment	37	350	11	--	350
Indexing:					
List	107	2400*	950	7500*	754
Vector	42	300	--	--	--
String	38	280	24	--	750
Graphics: **					
Circle:	5	***	--	--	12
Square 1:	9	***	--	--	31
Square 2:	11	***	--	--	32
PolySpiral 1:	10	***	--	--	20
PolySpiral 2:	30	***	--	--	Stack Overflow

Notes: The test results are in seconds.
Each test was done using a 5000 iterations.
Each index test accessed the 100th element of a 120 element object.
* Computed time based on 500 iterations.
** Done with only one iteration.
*** Line and point functions only supported. No turtle graphics.
-- Test not done since the operation was not supported.

Benchmark Results
Table 1

do not buy half a megabyte of RAM if you are going to use muLisp-82 only because you will not need it.

IQ Lisp is the best implementation for PC-DOS, which is its only operating system. There is no 8-bit counterpart. It is full of features, reasonably fast, and can utilize all the RAM you can put into your PC. Its error handling is superb and the function key and window support extremely useful. Although its software floating point support is very slow, IQ Lisp does support the 8087. The graphics support is merely adequate. The worst deficiency of the package is the lack of a resident screen editor.

TLC-Lisp is to CP/M-86 as IQ Lisp is to PC-DOS. Actually, TLC-Lisp offers a number of different choices, since it will also run under Concurrent CP/M-86 (a multitasking system that supports multiple windows). TLC-Lisp also offers an alternative to muLisp-82 under CP/M-86 for all the non-IBM PC compatible machines. These machines may actually offer a better development environment with faster (8Mhz) 8086/80186 based processors.

Although TLC-Lisp does not have infinite precision integers it does support a wider variety of data types and objects than any other implementation. Its screen editor is far superior to that of muLisp-82 and its graphics support on the IBM PC is more flexible than that of IQ Lisp. In fact, the TLC-Lisp turtle graphics rivals or exceeds that of most Logo implementations. Support of MACRO's, error handling, and 8087 support make TLC-Lisp an excellent choice, and its ability to utilize large amounts of memory is a definite plus. For those using Concurrent CP/M-86, you can restrict this feature.

micro-Prolog is in a class by itself. It does have an 8-bit version and runs under PC-DOS and MS-DOS, thus opening a gate for a number of machines into the Prolog world. Although the benchmarks do not show micro-Prolog as a speed demon, it is very good when it comes to pattern matching and database search. Its only major deficiency is its limited memory utilization, which prevents development of large applications. The lack of file update support may also be a serious problem in some applications.

In terms of Lisp, IQ Lisp and TLC-Lisp take the cake under PC-DOS and CP/M-86 respectively. The choice as to which system is better will be more difficult should either appear on the other operating system. muLisp-82 is a good alternative to either, if you have to move between operating systems. micro-Prolog wins hands down for Prolog implementations because it is the **only** implementation!

Lisp and Prolog have been used on mainframe computers for serious AI work. The appearance of these languages on the 8-bit microcomputers served as a basis for introducing a huge new group of people to AI methods and tools, because of the low cost of the systems and software. The new 16-bit implementations reviewed here fill the gap between the very high end mainframes and the low end micros. Significant applications can now be developed using sophisticated Lisp and Prolog tools on relatively inexpensive microcomputers.

LISP SOFTWARE SUPPLIERS

(muLISP-83)
The Soft Warehouse
P.O. Box 11174
Honolulu, HI 96828
(808) 734-5801 Price: $250
(please call after noon P.S.T.)

IQ Lisp
Integral Quality
P.O. Box 31970
Seattle, WA 98103-0070
(206) 527-2918 Price: $175

TLC-Lisp
The Lisp Company
330 C Village Lane
Los Gatos, CA 95030
(408) 354-3668 Price: Call

micro-Prolog
Prolog Systems
31 Crescent Drive
Milford, CT 06460
(203) 877-7988 Price: $275

TLC Lisp Benchmark Programs

Empty Loop Test:

```
(DE LOOP-TEST (FN LIMIT)
    (DO ((I 1 (+ I 1)))
        (((GE I LIMIT)))
        (FN) ) )
(LOOP-TEST () 5000)
```

CONS Test:

```
(DE X () (CONS A A))
(SETQ A '(1))
(LOOP-TEST X 5000)
```

Integer addition:

```
(DE ADD-TEST () (+ A B))
(SETQ A 1  B 2)
(LOOP-TEST ADD-TEST 5000)
```

Float addition:

```
uses ADD-TEST
(SETQ A 1.2  B 234324.3)
(LOOP-TEST ADD-TEST 5000)
```

Integer multiplication:

```
(DE MULTIPLY-TEST () (* A B))
(SETQ A 1  B 2)
(LOOP-TEST MULTIPLY-TEST 5000)
```

Float multiplication:

```
uses MULTIPLY-TEST
(SETQ A 1.2  B 234324.3)
(LOOP-TEST MULTIPLY-TEST 5000)
```

Assignment:

```
(DE ASSIGNMENT-TEST () (SETQ C C))
(SETQ C '(1 2 3))
(LOOP-TEST ASSIGNMENT-TEST 5000)
```

List Indexing:

```
(DE LIST-INDEX () (NTH L 100))
(SETQ L '( 1 2 3 4 ... 120))
(LOOP-TEST LIST-INDEX 5000)
```

Vector Indexing:

```
(DE VECTOR-INDEX () (V 100))
(SETQ V (NEWVEC 120))
(LOOP-TEST VECTOR-INDEX 5000)
```

String Indexing:

```
(DE STRING-INDEX () (NTH S 100))
(SETQ S "0123456789 .... 89")
(LOOP-TEST STRING-INDEX 5000)
```

Graphics Circle:

```
(LOOP 360 (FD 1) (TR 1))
```

Graphics Square 1:

```
(DE SQUARE (SIZE)
           (LOOP 4 (FD SIZE)
                   (TR 90) ) )
(DE SQUARE-TEST-1 ()
          (SQUARE 100) )
(LOOP-TEST SQUARE-TEST-1 100)
```

Graphic Square 2:

```
(LOOP 100
      (SQUARE 100)
      (TR 3) )
```

Graphic PolySpiral 1:

```
(DE POLY-SPIRAL (D A C)
    (IF (ZEROP C)
        ()
        (FD D)
        (TR A)
        (POLY-SPIRAL (+ D 1) (+ A 1) (- C 1))))
(POLY-SPIRAL 10 70 100)
```

Graphic PolySpiral 2:

```
(POLY-SPIRAL 10 70 200)
```

IQ Lisp Benchmark Programs

Empty Loop Test:

```
(DEFUN LOOP-TEST
       (LAMBDA (FN LIMIT)
               (LOOP (INITIAL I)
               (NEXT I (ADD1 I))
               (UNTIL (GE I LIMIT))
               (DO (FN)) ) )
(LOOP-TEST () 5000)
```

CONS Test:

```
(DEFUN X (LAMBDA () (CONS A A)))
(SETQ A (QUOTE (1)))
(LOOP-TEST X 5000)
```

Integer addition:

```
(DEFUN ADD-TEST (LAMBDA () (+ A B)))
(SETQ A 1)  (SETQ B 2)
(LOOP-TEST ADD-TEST 5000)
```

Float addition:

```
uses ADD-TEST
(SETQ A 1.2)  (SETQ B 234324.3)
(LOOP-TEST ADD-TEST 5000)
```

Integer multiplication:

```
(DEFUN MULTIPLY-TEST (LAMBDA () (* A B)))
(SETQ A 1)  (SETQ B 2)
(LOOP-TEST MULTIPLY-TEST 5000)
```

Float multiplication:

```
uses MULTIPLY-TEST
(SETQ A 1.2)  (SETQ B 234324.3)
(LOOP-TEST MULTIPLY-TEST 5000)
```

Assignment:

```
(DEFUN ASSIGNMENT-TEST () (SETQ C C))
(SETQ C  (QUOTE (1 2 3)))
(LOOP-TEST ASSIGNMENT-TEST 5000)
```

List Indexing:

```
(DEFUN NTH
       (LAMBDA (L N I)
               (SETQ I 1)
               (LOOP (INITIAL I 1)
                     (NEXT I (ADD1 I))
                     (UNTIL (GE I N))
                     (DO (SETQ L (CDR L)))
```

```
                                (RESULT (CAR L)) ) ) )
                (DEFUN LIST-INDEX (LAMBDA () (NTH L 100)))
                (SETQ L (QUOTE ( 1 2 3 4 ... 120)))
                (LOOP-TEST LIST-INDEX 5000)
```

Vector Indexing:
```
                (DEFUN VECTOR-INDEX (LAMBDA () (V 100)))
                (SETQ V (ARRAY 5 120))
                (LOOP-TEST VECTOR-INDEX 5000)
```

String Indexing:
```
                (DEFUN STRING-INDEX
                      (LAMBDA () (SUBSTRING S 100 100)))
                (SETQ S "0123456789 .... 89")
                (LOOP-TEST STRING-INDEX 5000)
```

Graphics: Dot and line functions are supported. Turtle graphics are not supported but could be implemented using existing functions. Not test programs were developed.

muLisp Benchmark Programs

Empty Loop Test:
```
                (DEFUN LOOP-TEST
                      (LAMBDA (FN LIMIT I)
                            (SETQ I 1)
                            (LOOP ((GREATERP I LIMIT))
                                  (SETQ I (ADD1 I))
                                  (FN) ) ) )
                (LOOP-TEST () 5000)
```

CONS Test:
```
                (DEFUN X (LAMBDA () (CONS A A)))
                (SETQ A (QUOTE (1)))
                (LOOP-TEST X 5000)
```

Integer addition:
```
                (DEFUN ADD-TEST (LAMBDA () (PLUS A B)))
                (SETQ A 1)  (SETQ B 2)
                (LOOP-TEST ADD-TEST 5000)
```

Float addition: not supported

Integer multiplication:
```
                (DEFUN MULTIPLY-TEST (LAMBDA ()(TIMES A B)))
                (SETQ A 1)  (SETQ B 2)
                (LOOP-TEST MULTIPLY-TEST 5000)
```

Float multiplication: not supported

Assignment:
```
                (DEFUN ASSIGNMENT-TEST () (SETQ C C))
                (SETQ C  (QUOTE (1 2 3)))
                (LOOP-TEST ASSIGNMENT-TEST 5000)
```

List Indexing:
```
                (DEFUN NTH
                      (LAMBDA (L N I)
                            (SETQ I 1)
                            (LOOP ((GREATERP I N) (CAR L))
                                  (SETQ I (ADD1 I))
                                  (SETQ L (CDR L)) ) ) )
                (DEFUN LIST-INDEX (LAMBDA () (NTH L 100)))
                (SETQ L (QUOTE ( 1 2 3 4 ... 120)))
                (LOOP-TEST LIST-INDEX 5000)
```

Vector Indexing: not supported

String Indexing:
```
                (DEFUN STRING-INDEX
                      (LAMBDA () (SUBSTRING S 100 100)))
                (SETQ S (QUOTE A0123456789 .... 89))
                (LOOP-TEST STRING-INDEX 5000)
```

Graphics: Not supported

micro-Prolog Benchmark Programs

Empty Loop Test:
```
                ((LOOP-TEST X 1))
                ((LOOP-TEST X Y)
                     (SUM Z 1 Y)
                     X
                     /
                     (LOOP-TEST X Z) )
                ((EMPTY-TEST))
                ?((LOOP-TEST (EMPTY-TEST) 5000))
```

CONS Test:
```
                ((CONS-TEST) (EQ (X | Y) (1)))
                ?((LOOP-TEST (CONS-TEST) 5000))
```

Integer addition:
```
                ((ADD-TEST) (SUM 1 2 X))
                Note: access of constant, not variables
                ?((LOOP-TEST (ADD-TEST) 5000))
```

Float addition:
```
                ((FLOAT-ADD-TEST) (SUM 1.2 234342.3 X))
                Note: access of constant, not variables
                ?((LOOP-TEST (FLOAT-ADD-TEST) 5000))
```

Integer multiplication:
```
                ((MULTIPLY-TEST) (TIMES 1 2 X))
                Note: access of constant, not variables
                ?((LOOP-TEST (MULTIPLY-TEST) 5000))
```

Float multiplication:
```
          ((FLOAT-MULTIPLY-TEST) (TIMES 1.2 234324.3 X))
          Note: access of constant, not variables
          ?((LOOP-TEST (FLOAT-MULTIPLY-TEST) 5000))
```

Assignment: not done since conventional assignment is significantly different from adding clauses into the data base.

List Indexing:
```
                ((INDEX (X | Y) 1 X))
                ((INDEX (X | Y) Z X1)
                        (SUM X2 1 Z)
                        /
                        (INDEX Y X2 X1))
                ((A | 0 1 2 3 4 5 6 7 8 9 ... 8 9))
                ((B X) (INDEX X 100 Y))
                ?((A | X)
                  (EQ Y (B X))
                  (LOOP-TEST Y 5000) )
```

```
Vector Indexing:    not supported

String Indexing:    strings over 60 characters not supported

Graphics:           Not supported
```

DR Logo Benchmark Programs

```
Empty Loop Test:    TO LOOP.TEST :FN :TIMES
                      REPEAT :TIMES [RUN :FN]
                    END
```

As we go to press

New versions of muLISP and TLC Lisp have been released with major improvements since the original work for this review was done. The following comments take these new versions into account.

The latest version is muLISP-83 which has a number of major enhancements. The documentation has been redone using the Lisp syntax to formally define functions. More examples and comments have been added which make the documentation very good. A Maclisp and Interlisp compatability function library has been added to assist in software migration between other systems. An interactive, online introduction for novice users is included. This is a very good way to learn Lisp.

A number of enhancements and corrections have been added to the new implementation. Assembly language routines can be added along with a table driven scanner/reader which includes various types of read macros. Error handling has been vastly improved and the CATCH/THROW constructs have been added. Even random file I/O is included. The maximum integer precision is now limited to 64 kbytes per number (which is large by any measure).

Although muLISP does not support windows or graphics directly, lesson #6 of the tutorial shows how to implement basic turtle graphics using DOS interrupts and ROM routines in the IBM PC.

The memory allocation is slightly different than described in the article. List space is limited to 128 kbytes (32K list cells) while D-code resides in a separate area which can be up to 60 kbytes. Garbage collection speed has been improved.

TLC-Lisp also has a number of improvements including new documentation and a real compiler. The compiler typically doubles the speed and cuts the code size from thirty to fifty percent. The turtle graphics drawing speed has also doubled.

The class system has been improved and so has the resident screen editor. In fact, the compiler can be invoked directly from the editor with two keystrokes. Full PC-DOS 2.x file access is supported including subdirectory support.

Versions are available for PC-DOS, MS-DOS and CP/M-86. Graphics are available for the IBM PC and compatible machines.

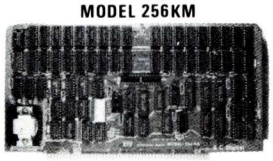
A new version of micro-Prolog has also been released which supports an assembly language interface. It also comes with a DEC-10 ProLog emulation mode.

```
                    TO DO.NOTHING
                    END
                    LOOP.TEST [DO.NOTHING] 5000

CONS Test:          TO CONS.TEST
                       (LOCAL "T)
                       MAKE "T FPUT :A :A
                    END
                    MAKE "A [1]
                    LOOP.TEST [CONS.TEST] 5000

Integer addition:   TO ADD.TEST
                       (LOCAL "T)
                       MAKE "T :A + :B
                    END
                    MAKE "A 1
                    MAKE "B 2
                    LOOP.TEST [ADD.TEST] 5000

Float addition:     MAKE "A 1.2
                    MAKE "B 234324.3
                    LOOP.TEST [ADD.TEST] 5000

Integer multiplication:
                    TO MULTIPLY.TEST
                       (LOCAL "T)
                       MAKE "T :A * :B
                    END
                    MAKE "A 1
                    MAKE "B 2
                    LOOP.TEST [MULTIPLY.TEST] 5000

Float multiplication:
                    MAKE "A 1.2
                    MAKE "B 234324.3
                    LOOP.TEST [MULTIPLY.TEST] 5000
Assignment:         TO ASSIGNMENT.TEST
                       MAKE "C :C
                    END
                    MAKE "C [1 2 3]
                    LOOP.TEST [ASSIGNMENT.TEST] 5000
```

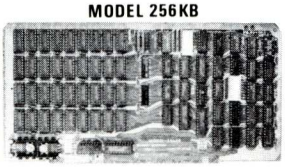
```
List Indexing:      TO LIST.INDEX.TEST
                       (LOCAL "T)
                       MAKE "T ITEM 100 A
                    END
                    MAKE "A [0 1 2 3 4 5 6 7 8 9 ... 8 9]
                    LOOP.TEST [LIST.INDEX.TEST] 5000

Vector Indexing:    not supported

String Indexing:    TO STRING.INDEX.TEST
                       (LOCAL "T)
                       MAKE "T ITEM 100 A
                    END
                    MAKE "A "A0123456789 .... 89
                    LOOP.TEST [STRING.INDEX.TEST] 5000

Graphics Circle:    REPEAT 360 [FD 1 RT 1]

Graphics Square 1:  TO SQUARE :SIZE
                       REPEAT 4 [FD :SIZE RT 90]
                    END
                    TO SQUARE.TEST.1
                       SQUARE 100
                    END
                    LOOP.TEST [SQUARE.TEST.1] 100

Graphic Square 2:   REPEAT 100 [SQUARE.TEST.1 RT 3]

Graphic PolySpiral 1:
                    TO POLY-SPIRAL :D :A :C
                       TEST :C = 0
                       IFFALSE [FD :D
                                RT :A
                                POLY.SPIRAL :D + 1
                                            :A + 1
                                            :C - 1]
                    END
                    POLY.SPIRAL 10 70 100

Graphic PolySpiral 2:
                    POLY.SPIRAL 10 70 200
```

For S100 bus by S. C. Digital, Inc.

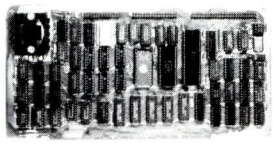
MODEL 256KM



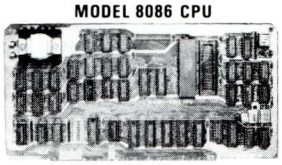
MODEL 256KB



MODEL FDC-1



MODEL 8086 CPU



256 K/1M DYNAMIC RAM Board Model 256KM $795

● 256K/1M bytes using 64K or 256K DRAMs ● 8/16b data ● 24b Address ● Parity per byte ● 175nsec access time ● will run Z80/Z8000 to 6mhz, 8086, 80186, 68000 to 8mhz without wait states ● transparent refresh, unlimitted DMA ● with 1MB using 256K drams.

256K/1M DYNAMIC RAM BOARD Model 256KB-256 $459

● 256K/1M bytes using 64K or 256K DRAMs ● 8b data ● 16 or 24b address ● parity per byte ● Memory mapping in 16K blocks ● 175 nsec access time ● Addressable in 128k, 192k, or 256k boundaries, compatible with Z-100* systems ● with 256KB using 256K drams, expandable to 1 mega byte, less memory mapping ● add $20 for memory mapping.

FLOPPY DISK CONTROLLER Board Model FDC-1 $375

● Single or double sensity, sides, in any combination of up to four 8" or 5¼" drives ● Digital phase-locked loop ● DMA data transfer with cross 64K bountaries, 24b address, DMA arbitration ● built in monitor/boot EPROM that accomodates two different processors ● serial port to 19.2Kbaud ● uses 765A/8272 ● with CPM bios programs

8086 CPU Board Model 8086 CPU $375

● 8/4 mhz SW selectable ● 8087 interface ● provision to fun two processors on a bus such as our Z80 CPU ● convertable to 10, 12mhz clock ● optimized for DRAM boards.

80286 CPU Board Model 80286 from $350

● 8/4 mhz switchable ● 80827 interface ● provision to run two processors on a bus ● convertable to 8mhz ● separate built in colck for 80287 ● optimized for DRAM boards.

SUPPORT Board Model Support-1 $395

● 4 serial, full handshakes, two with software programmable baud rates ● Centronics ● SASI interface ● Real/interval times ● Calendar-clock with battery backup ● expandable interrupt controllers for 8086 or 8080/Z80 ● CPU switching to run 2 processors on a bus such as our 8086 or 80286 and Z80 CPU boards.

I/O Interface board Model 3SPC-N $229

● 3 serial RS-232C with switch selectable baud rates, 110 to 19.2kbaud. ● 1 parallel.

Board Sets: 8086, 256KM (with 512KB), FDC-1, 3SPC-N $1388
Z80, 256KB-256, FDC-1, 3spc-N $1150

S-100 COMPUTER 'System 16' $3200

8086 based at 8mhz, with 512kb, 5 serial ports, 1 centronics, 1 SASI, battery backed calendar clock, real time clock, interrupt driven, 10 slot card cage, two 5.25" floppy drives with 500 kb transfer rates and 1.2 mbyte storage each, with CPM 86 operating system (Concurrent Dos available soon). Cabinet has room for full size 5¼" hard disk.

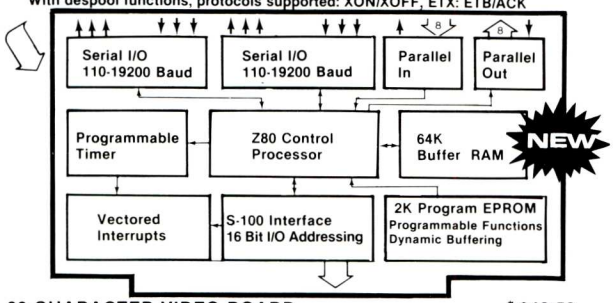
Operating Systems available: CPM 2.2, CPM 3.0, CPM 86, MSDOS.

*CPM is registered trade mark of Digital Research Inc. Z-100 is registered trade mark of Zeith Corporation.

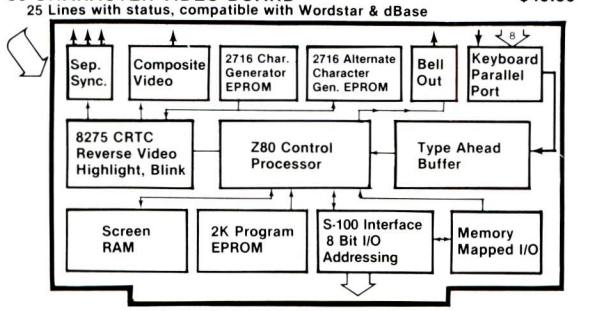
Please call for latest prices.
Prices subject to change without notice.

S.C. DIGITAL, INC.
1240 N. Highland Ave., Suite 4 ● P.O. Box 906, Aurora, Illinois 60507
Phone: (312) 897-7749

BUFFERED I/O BOARD Introductory Price *$59.95
With despool functions, protocols supported: XON/XOFF, ETX: ETB/ACK



80 CHARACTER VIDEO BOARD *$49.50
25 Lines with status, compatible with Wordstar & dBase



Includes Bareboard, Heatsink & Documentation Call or write for more information.

Simpliway Products Co.
P.O. Box 601
Hoffman Estates, IL. 60195
(312) 359-7337

SPC

VISA

MasterCard

OEM dealer pricing available, $3.00 S/H, IL. Res. add 7% tax
dBase™ - of Ashton - Tate Corp. — Wordstar™ - of Micropro Int'l. Corp

NEW PRODUCTS

SERIAL ADAPTER FOR THE VDB-A2-

- Plugs into keyboard socket
- Direct connect to Keytronic (IBM) serial ASCII keyboard
- Internal speaker for BEEPER
- Tx port for non VDB-A2 use
- Hardware programmed Rx & Tx
- Dual baud-rate generator
- Bare-board OR assembled

DOUBLE-HEIGHT S-100 PROTOTYPING-

- Oversized heatsinks for HI-CURRENT reg's: +5, +/−12 V.
- Use with wire-wrap sockets OR direct solder connections
- GOLD plated edge connector
- NO PLATING CUTS required
- NO wasted address decoding

WATCH OUR ADS FOR OTHER PRODUCTS COMING SOON

- MODEM ADAPTER FOR THE BIO-1
- 256K RAM SOFTWARE FOR BIO-1
- HIGH FEATURE DISC CONTROLLER
- Z80 C.P.U. / MEMORY MANAGER
- 1 MEGABYTE DRAM BOARD

Specifications subject to change without notice

Loadable BIOS Drivers For CP/M

by Cal Sondgeroth

Add or change BIOS code at boot time without changing the BIOS code itself.

Introduction

One reason that CP/M is such a popular operating system for small computers is its interaction with a user provided I/O system. This standard interface to the computer hardware means you can run CP/M on anything that executes the 8080 instruction set.

In the early days when floppy disks were all 8 inch single density, a CP/M BIOS was a fairly simple piece of code. About all you needed were routines to read and write a disk sector and interact with one or more serial lines for I/O. With the coming of double density (and double sided) disks and mini-floppies and Winchesters, the complexity of the BIOS has become such that it is usually supplied as an integral part of a pre-configured computer system.

As a result, a great many CP/M users tend to shy away from changing their BIOS code; and for good reason. Since it is so complex, it is often difficult to make changes without introducing disastrous side effects. There are many cases, however, where a modified BIOS handler for an I/O device can enhance a specific application.

No matter how complex a BIOS may be, it's basic interface to the CP/M operating sytem remains the same. The same I/O subroutines are provided, and their entry points are defined by a fixed jump table. This article presents some ideas on how to provide your system with what might be called "loadable drivers" by using these fixed entry points. The more critical code (like disk I/O) can be left unchanged while new handlers for other things (like the console, list, reader, etc.) can be modified to suit an application.

Using the methods described here, you can write and test new BIOS handlers on an individual basis and run the new system without modifying a line of the original BIOS source. In fact, if you know how to interact with the I/O devices you want to control, you don't even have to have the source code for your BIOS.

Every time you do a cold start, your machine is back to using the original vendor supplied BIOS until you load one or more of your new drivers for special applications. This means that you can debug a device handler by making changes, re-compiling and testing without all the hassle of re-compiling an entire BIOS, patching it into CP/M and doing a system generation.

The BIOS entry point jump table

At the start of every BIOS, there is a table of 17 jump instructions. These jumps are placed in a fixed order and each one goes to the start of the internal code that handles a specific I/O function. Since CP/M always knows where the BIOS starts, it can access the device handlers by referencing these absolute address jumps with a call instruction. Each handler exits via a return to CP/M.

The jump table looks like Figure 1.

Looking at the jump table in Figure 1, you can see that in order to provide a different handler for one of the BIOS routines, all you need to do is get the

```
Offset  Jump Instruction      Description of handler routine
------  ----------------  ------------------------------------
+0000     JMP COLD          ;COLD START INITIALIZATION
+0003     JMP WBOOT         ;WARM BOOT OPERATION
+0006     JMP CONST         ;CONSOLE CHARACTER READY STATUS
+0009     JMP CONIN         ;READ CONSOLE INPUT CHARACTER
+000C     JMP CONOUT        ;SEND C-REGISTER TO CONSOLE
+000F     JMP LIST          ;SEND C-REGISTER TO LIST DEVICE
+0012     JMP PUNCH         ;SEND C-REGISTER TO THE PUNCH
+0015     JMP READER        ;RETURN NEXT READER CHARACTER
+0018     JMP HOME          ;DISK SEEK TO TRACK 00
+001B     JMP SELDSK        ;SELECT DISK IN C-REGISTER
+001E     JMP SETTRK        ;SET TRACK FROM BC REGISTERS
+0021     JMP SETSEC        ;SET SECTOR FROM BC REGISTERS
+0024     JMP SETDMA        ;SET DISK I/O ADDRESS (BC)
+0027     JMP READ          ;READ TRACK AND SECTOR
+002A     JMP WRITE         ;WRITE TRACK AND SECTOR
+002D     JMP LISTS         ;RETURN LIST DEVICE STATUS
+0030     JMP SECTRAN       ;CONVERT LOGICAL TO PHYSICAL SECTOR
```

Figure 1 - CP/M BIOS Entry Point Jump Table

new handler in memory somewhere and change the table jump so it goes to the new code rather than the original code inside the BIOS. Of course this assumes that your BIOS is in RAM memory. If your computer has the BIOS permanently coded in ROM, the ideas presented here are not of much use.

Finding a place to put the new I/O driver can be a problem if your version of CP/M uses all of memory as is usually the case. Unless there is some unused memory at the top of the BIOS, you are out of luck.

A possible solution is to regenerate CP/M for a smaller memory size. That would leave some space above the BIOS for your new driver. The disadvantage of this of course is that the memory space is always used whether you have a BIOS extension or not. The relocatable drivers described here allow you to use the maximum memory space on your machine reserving only the amount necessary to contain the new driver when you install it.

If you can find some memory to use and want to try modifying one of the BIOS routines, you can write the new handler with an absolute origin and assemble it. Then write a CP/M program that loads the new code and changes the BIOS jump accordingly.

Using the transient program area for driver code

If we look at the CP/M memory map (see the left hand side of Figure 2) there is a huge section of memory that is used for processes that run as normal programs. If there were some way to put our new driver code up in the high part of this memory, and then do something so it would not get destroyed by other programs, we could run the driver code there with only a small impact on the total memory.

Most CP/M programs use everything from 0100H right on up to the base of the CP/M BDOS. That is they can overwrite the Console Command Processor since it is reloaded when the program exits via a warm boot operation. Such a program could destroy any driver code placed at the top of the TPA.

There is a way out, however. All CP/M transient programs are written so they automatically adjust themselves to use the maximum amount of memory available. In these days of cheap memory it is sometimes easy to forget that you don't have to have 64K to run alot of CP/M stuff. Indeed, the standard V2.2 system is set up for a 20K machine — lots of memory in 1979.

To find out how much memory is available to it, a program examines the target address of the jump instruction at location 0005H; the constant system call entry point. Since this jump goes to the start of the BDOS, its target address is the lowest address needed by CP/M to run the application. The CCP can be destroyed since it plays no part in transient program operation.

It would seem that if we modified the location 0005H jump so it went somewhere below the base of the BDOS, we could allocate some memory (like for the drivers we want to add) and it would not be overwritten by other programs. All we would need then is another jump instruction at the beginning of the allocated memory that continues on to the real BDOS entry so calls to location 0005H would reach it like always. Since CP/M programs would find the lower target address in the jump at 0005H they would think they were running in a smaller size system and adjust their memory useage accordingly.

To make sure the memory remains protected, there is one more thing that needs to be done. Since a warm boot usually reloads all of the operating system from the start of the Console Command Processor up to the start of the BIOS and resets the location 0005H jump as part of the process we have to supply a new warm boot routine for the BIOS. Otherwise, the location 0005H jump would be set back to normal on the first warm boot after we tried to allocate some memory, and the next program to run would not know about it. The sample BIOS extension below provides a warm boot handler that takes care of this.

Since the warm boot no longer reloads the operating system, the CCP is kept in memory all the time so it is available to process command lines. Memory for the loadable drivers is set up just below the base address of the CCP.

The right hand side of Figure 2 shows a memory map of CP/M with a driver module installed just below the CCP. Note how the jump instruction at 0005H in that map goes to the start of the driver. The driver starts its code with a jump instruction that continues on to the normal BDOS entry point. We will see later how all of this code gets loaded and the jump instructions are changed so they go to the right places.

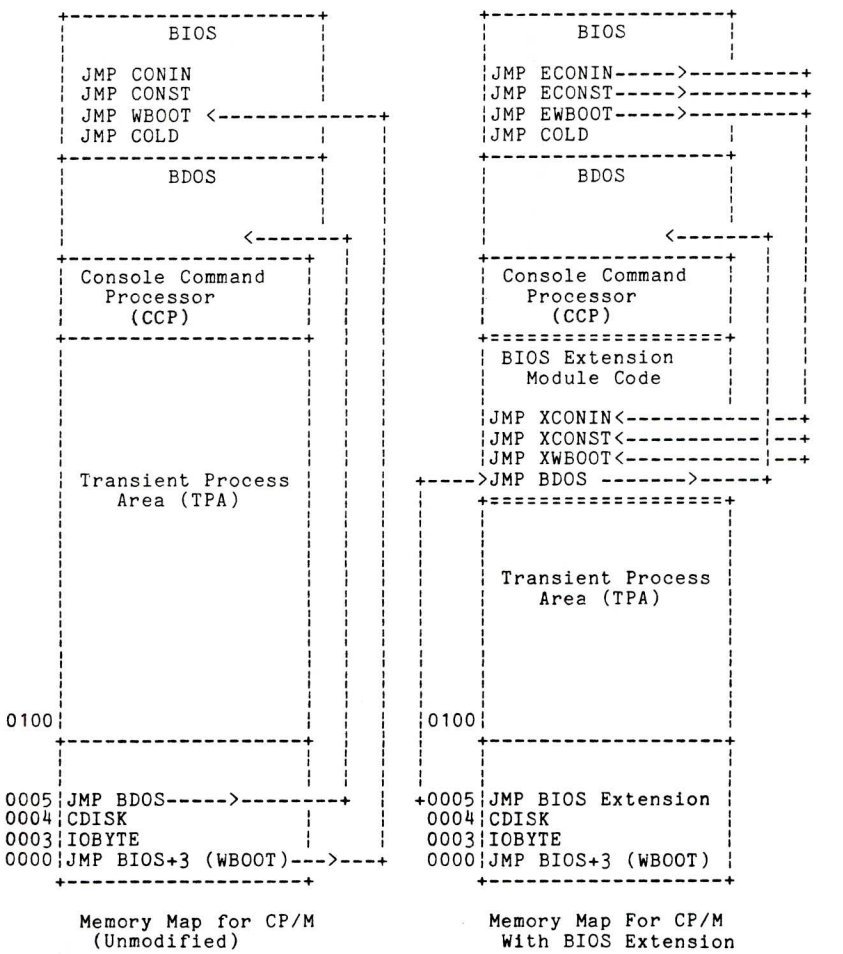


Figure 2 - CP/M Memory Maps

Getting the driver code installed

The example BIOS extension and the relocatable loader program make use of the Digital Research RMAC relocatable assembler and LINK linking loader. They allow you to write a BIOS driver and load and relocate it to run in any size CP/M system putting it just below the CCP no matter where that happens to be. If you don't have access to RMAC and LINK you can write your driver so it is origined at a fixed address below the CCP. Then you will need to write an absolute loader program to install it and fix the jump instructions.

Using the relocating loader is much cleaner. It automatically allocates just enough memory to hold the driver and is not bound to any absolute addressing; the extension is automatically loaded and relocated to run at its new address.

System page relocatable files

The Digital Research LINK utility has an output option that produces this type of file rather than an absolute COM file that runs in the TPA. A system page relocatable file has a type of "SPR" and contains object code that is origined at a base address of 0000H. It also has a map that indicates which bytes of the code have to be adjusted if they are moved so the module base address is other than 0000H. The only restriction on relocating the code and running it elsewhere is that it be moved to an even page boundary; i.e., starting somewhere at XX00H, where XX is the high byte of the new address. This is so because the relocation map only indicates that a high byte of the code is to be adjusted when it is moved; the low bytes remain the same as they were when it was origined at 0000H.

A dump of the system page relocatable file for the sample BIOS extension module is shown in Figure 3. The first 256 bytes contain information about the sizes of the various sections of the program. For our purposes, we will only be concered with the size of the code section. This is found in the 2nd and 3rd bytes of the first page (256 bytes) of the file. In this case byte 2 (the low part) is 0A3H and byte 3 (the high part) is 00H, so the number of bytes of object code in the file is 00A3H, or 163 bytes.

The actual object code begins in the 2nd 256 byte page of the file, starting at address 0200 in Figure 3. You can see the jump table at the beginning of the BIOS extension module there.

The thing that makes relocation of the object code possible is the bit map that starts at the first byte after the object code. In Figure 3 it is at address 02A3 (the start of the object code plus its size). The map is set up so that when one takes each byte and shifts the bits

```
DDT VERS 2.2
-IXBIOS.SPR
-R
NEXT  PC
0300 0100
-D100,2FF
0100 00 A3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...
01D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0200 C3 00 00 C3 66 00 C3 39 00 C3 3C 00 C3 3F 00 C3 ....f..9..<..?..
0210 42 00 C3 45 00 C3 48 00 C3 4B 00 C3 4E 00 C3 51 B..E..H..K..N..Q
0220 00 C3 54 00 C3 57 00 C3 5A 00 C3 5D 00 C3 60 00 ..T..W..Z..]..`.
0230 C3 63 00 C3 00 00 C3 00 00 C3 00 00 C3 00 00 C3 .c..............
0240 00 00 C3 00 00 C3 00 00 C3 00 00 C3 00 00 C3 00 ................
0250 00 C3 00 00 C3 00 00 C3 00 00 C3 00 00 C3 00 00 ................
0260 C3 00 00 C3 00 00 31 80 00 01 80 00 CD 57 00 21 ......1......W.!
0270 00 00 22 06 00 11 89 00 0E 09 CD 00 00 2A 01 00 ..".........*..
0280 11 00 EA 19 3A 04 00 4F E9 0D 0A 42 49 4F 53 20 ....:..O...BIOS
0290 45 58 54 45 4E 53 49 4F 4E 20 49 4E 20 50 4C 41 EXTENSION IN PLA
02A0 43 45 24 04 92 49 24 92 49 20 00 00 00 00 00 00 CE$..I$.I ......
02B0 02 41 08 00 00 00 00 00 1A 1A 1A 1A 1A 1A 1A 1A .A..............
02C0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
02D0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
02E0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
02F0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
```

CODE SIZE

OBJECT CODE

RELOCATION BIT MAP

Figure 3 - XBIOS.SPR File Dump

```
;*****************************************************
;*                                                   *
;*                  X   B   I   O   S                *
;*                                                   *
;*****************************************************
;*                                                   *
;*  A LOADABLE BIOS DRIVER MODULE FOR CP/M VERSION 2.2
;*                                                   *
;*                BY: CAL SONDGEROTH                 *
;*                    800 FIFTH AVENUE               *
;*                    MENDOTA, IL.  61342            *
;*                                                   *
;*****************************************************
;* REV |  DATE   |             DESCRIPTION            *
;*-----+---------+------------------------------------*
;* 1.0 | 02JAN84 | INITIAL LOADABLE BIOS MODULE CODE  *
;*****************************************************

        TITLE 'LOADABLE BIOS MODULE FOR CP/M V2.2'
        PAGE 58

CR      EQU 0DH                 ;ASCII CARRIAGE RETURN CHARACTER
LF      EQU 0AH                 ;ASCII LINE FEED CHARACTER
CDISK   EQU 0004H               ;CP/M DEFAULT DISK VARIABLE
PRINTF  EQU 9                   ;BDOS PRINT STRING FUNCTION

; THE FOLLOWING JUMP TABLE IS USED TO DEFINE NEW ENTRY POINTS
; FOR THE BIOS ROUTINES THAT ARE TO BE MODIFIED.  ANY ROUTINES
; THAT ARE NOT PART OF THE LOADABLE BIOS JUMP TO THE ORIGINAL
; BIOS SUBROUTINE ENTRY POINTS.

BDOS    JMP $-$             ;CONTINUATION JUMP TO CP/M BDOS
        JMP XWBOOT          ;WARM BOOT ROUTINE FOR LOADABLE BIOS
        JMP CONST
        JMP CONIN
        JMP CONOUT
        JMP LIST
        JMP PUNCH
        JMP READER
        JMP HOME
        JMP SELDSK
        JMP SETTRK
        JMP SETSEC
        JMP SETDMA
        JMP READ
```

```
        JMP WRITE
        JMP LISTS
        JMP SECTRAN
        PAGE
; WHEN THE LOADABLE BIOS IS FIRST INSTALLED IN MEMORY, THE
; CONTENTS OF THE ORIGINAL CP/M BIOS JUMP TABLE ARE COPIED
; INTO THE FOLLOWING TABLE IN THE LOADABLE BIOS.  THESE JUMPS
; PRESERVE THE ENTRY POINTS FOR ANY BIOS HANDLERS THAT ARE
; NOT GOING TO BE CHANGED.

COLD    JMP $-$
WBOOT   JMP $-$
CONST   JMP $-$
CONIN   JMP $-$
CONOUT  JMP $-$
LIST    JMP $-$
PUNCH   JMP $-$
READER  JMP $-$
HOME    JMP $-$
SELDSK  JMP $-$
SETTRK  JMP $-$
SETSEC  JMP $-$
SETDMA  JMP $-$
READ    JMP $-$
WRITE   JMP $-$
LISTS   JMP $-$
SECTRAN JMP $-$

;*************************************************************
; THE FOLLOWING ENTRY POINT IS USED TO HANDLE THE CP/M WARM
; BOOT WHEN A LOADABLE BIOS MODULE IS IN USE.  THIS ROUTINE
; SHOULD ALWAYS BE INCLUDED IN A LOADABLE BIOS.  IT BYPASSES
; THE RELOAD OF THE OPERATING SYSTEM AND REINITIALIZATION OF
; THE LOW MEMORY JUMP VECTORS THAT ARE PART OF A NORMAL CP/M
; VERSION 2.2 WARM BOOT.
;*************************************************************

XWBOOT  EQU $
        LXI SP,0080H            ;RESET THE STACK POINTER
        LXI B,0080H             ;AND SET DMA ADDRESS TO
        CALL SETDMA             ;GO TO CP/M DEFAULT I/O AREA
        LXI H,BDOS              ;RESET THE LOCATION 0005H JUMP
        SHLD 0006H              ;TO GO TO OUR LOADABLE BIOS CODE
        LXI D,WBMSG             ;OUTPUT A MESSAGE TO INDICATE
        MVI C,PRINTF            ;LOADABLE BIOS IS IN PLACE
        CALL BDOS
        LHLD 0001H              ;FETCH BIOS+3 ADDRESS AND THEN
        LXI D,-1600H            ;COMPUTE THE ADDRESS OF THE
        DAD D                   ;CONSOLE COMMAND PROCESSOR PLUS 3
        LDA CDISK ! MOV C,A     ;FETCH CURRENT DISK
        PCHL                    ;AND VECTOR INTO THE CCP

WBMSG   DB CR,LF,'BIOS EXTENSION IN PLACE$'



**********************************************************
*                                                        *
*                  X   L   O   A   D                     *
*                                                        *
**********************************************************
*                                                        *
*        BIOS EXTENSION LOADER TRANSIENT PROCESS         *
*                                                        *
*                BY: CAL SONDGEROTH                      *
*                    800 FIFTH AVENUE                    *
*                    MENDOTA, IL. 61342                  *
*                                                        *
**********************************************************
* REV |  DATE   |              DESCRIPTION                *
*-----+---------+-----------------------------------------*
* 1.0 | 02JAN84 | INITIAL XBIOS MODULE LOADER CODE        *
**********************************************************

        TITLE 'XLOAD - BIOS EXTENSION LOADER'
        PAGE 58

BOOT            EQU 0000H       ;WARM BOOT ENTRY
BDOS            EQU 0005H       ;DOS ENTRY ADDRESS
FCB             EQU 005CH       ;DEFAULT FILE BLOCK
CONOUT          EQU 2           ;CONSOLE OUTPUT FUNCTION
OPEN            EQU 15          ;BDOS OPEN FUNCTION
SETDMA          EQU 26          ;BDOS SET DMA ADDRESS
READ            EQU 20          ;BDOS READ SEQUENTIAL

        ORG 0100H               ;CP/M TRANSIENT PROCESS
XLOAD   EQU $
        LXI H,0 ! DAD SP        ;SAVE CP/M'S STACK POINTER
```

out to the left one at a time for each byte of object code, a one indicates the byte must be modified (relocated) by adding the page offset of the place where the code is to run. If a map bit is a zero, then the corresponding byte can be moved without modifying it.

So, to install a driver from its SPR file we move its object code to the desired location and use the bit map to go through and modify any bytes that are the high part of address dependent code. This whole process is very similar to the scheme of assembling a source file twice; first with an origin of 0100H and next with an origin of 0200H and comparing the bytes in the two hex files to determine which need to be relocated when the code is moved. The system page relocatable file output option of the LINK utility does this automatically producing the object code plus the relocation information.

Sample BIOS extension module

A source code listing (XBIOS) is included here. It is assembled with the RMAC relocatable assembler. There are no ORG statements; its code begins at 0000.

The sample module starts with a completely new jump table. These are the entry points that CP/M will use after the extension has been loaded and relocated.

Note the first jump. It is coded as "JMP $-$" without a specific address. This is the one that continues on to the BDOS as described above. The target of this jump gets filled in by the loader program (XLOAD) described below. XLOAD determines the size of the CP/M system and inserts the correct address. This first jump replaces the normal BIOS cold start entry which is not used after the initial power up reset.

The second jump goes to a routine called "XWBOOT". This is the handler that takes care of the warm boot operation when the loadable driver is present. It will be part of every BIOS extension you use, and it does everything except read the operating system from the disk system tracks. Note that it resets the location 0005H system entry jump, but to the base address of the driver module to maintain it in memory.

Since the sample extension does not have any code for modified handlers, the other jumps go to the original BIOS handlers, so they remain unchanged. Of course you would code the appropriate jumps to go to any new handlers that you include in an extension module. You can assemble and install the sample XBIOS. The only difference you will find is the "EXTENSION IN PLACE" message on each warm boot.

Also, you will probably notice that a warm boot goes faster since there is no disk read on the system tracks. When you write your own driver extensions, you can change the warm boot message to identify the driver.

The original BIOS jump table

In the sample extension, there is a second jump table immediately after the new jump table at the beginning. The loader program copies the existing BIOS jump table into this area. The purpose of that of course is to maintain access to all of the original BIOS routines. You will probably only be using one or two modifications, so you want to be able to access the other handlers and not disturb them. If your extensions need to refer to the original BIOS handlers, they can do so through the labels in this table.

After it is safely copied away, the loader program modifies the original BIOS jump table so its entries go to the jump table at the start of the extension module. Thus when CP/M calls the BIOS, it still accesses the same address, but the jump goes to the new BIOS jump table. In some cases, that will go on to the normal BIOS handler. If there is a new module in the extension, it will be used.

Building a BIOS extension

In order to get a loadable extension module for the sample XBIOS or one you write, first assemble it with the RMAC assembler:

A>RMAC XBIOS <CR>

That will produce a relocatable file XBIOS.REL. Then give XBIOS.REL to the LINK utility to produce a system page relocatable file that the loader program can use:

A>LINK XBIOS [OS] <CR>

where the "[OS]" option switch tells LINK to output an SPR file. When LINK finishes there will be a file XBIOS.SPR on the disk. It is instructive to load this file under DDT and examine it. You will find the code size, the object code and the bit map as described above and in Figure 3.

Relocating BIOS extension loader program

This is the program XLOAD referred to above. A listing for it is also shown. It has an absolute origin of 0100H and runs from a COM file. It can be assembled with either the ASM or MAC assemblers. Or you can assemble it with RMAC (by removing the ORG 0100H statement) and generate a COM

```
        SHLD STACK              ;AND CREATE A LOCAL STACK
        LXI SP,STACK            ;FOR THE TRANSIENT PROCESS
        LDA FCB+1 ! CPI ' '     ;MAKE SURE NEW BIOS FILE NAME
        JNZ XL20                ;WAS ENTERED ON THE INPUT
        CALL MSG
        DB 'XLOAD - ** NO LOADABLE BIOS NAME GIVEN **',0
CPM$EXIT:
        LHLD STACK ! SPHL
        RET

XL20:   LXI H,FCB+9             ;GET TO FCB FILE TYPE
        MVI M,'S' ! INX H       ;FORCE "SPR" FILE TYPE
        MVI M,'P' ! INX H       ;FOR DATA READ
        MVI M,'R' ! INX H
        MVI C,24                ;REMAINING FCB CHARACTER COUNT
SLCLR:  MVI M,0 ! INX H         ;CLEAR REST OF THE FCB
        DCR C ! JNZ SLCLR       ;WRITE ALL ZEROES IN FCB
        LXI D,FCB               ;GET TO FILE CONTROL BLOCK
        MVI C,OPEN              ;TRY TO OPEN THE SPR FILE
        CALL BDOS
        ANA A ! JP SL50         ;CONTINUE IF FILE OPENED
        CALL MSG
        DB 'XLOAD - ** CAN''T OPEN BIOS SPR FILE **',0
        JMP CPM$EXIT

; READ THE SPR FILE INTO MEMORY IN PREPARATION FOR MOVING IT UP
; AND RELOCATING IT TO RUN.

SL50:   LXI H,BUF               ;HL -> DMA ADDRESS
SL60:   PUSH H                  ;SAVE DMA ADDRESS PTR INSIDE LOOP
        XCHG                    ;DE = DMA ADDRESS POINTER
        MVI C,SETDMA            ;SET CP/M DISK I/O ADDRESS
        CALL BDOS               ;TO NEXT BUFFER SECTOR
        LXI D,FCB               ;POINT TO FILE CONTROL BLOCK
        MVI C,READ              ;READ FILE SEQUENTIAL
        CALL BDOS               ;INTO NEXT BUFFER SECTOR SPOT
        POP H                   ;RE-FETCH CURRENT DMA ADDRESS
        LXI D,128 ! DAD D       ;AND MAKE THE NEXT DMA ADDRESS
        ANA A ! JZ SL60         ;CONTINUE IF MORE TO READ

; THE FOLLOWING MOVES THE RELOCATABLE CODE FROM THE SPR FILE
; UP IN MEMORY JUST BELOW EITHER THE CP/M CONSOLE COMMAND
; PROCESSOR OR BELOW THE LAST SYSTEM MODIFICATION.

        DI                      ;DONT ALLOW INTERRUPTS HERE
        LDA BOOT+2 ! SUI 16H    ;COMPUTE PAGE ADRS OF CCP
        MOV H,A                 ;H = CCP BASE PAGE ADDRESS
        LDA BDOS+2              ;GET CURRENT BDOS PAGE ADDRESS
        CMP H                   ;TEST AGAINST CCP BASE PAGE ADRS
        JNC REL1                ;IF SYSTEM NOT MODIFIED
        LHLD BDOS+1             ;ELSE GET CURRENT TOP OF MEMORY
REL1:   MVI L,0                 ;GET ON AN EVEN PAGE BOUNDARY
        XCHG                    ;DE THEN = LOWEST USED MEMORY
        LXI H,BUF+1             ;FETCH POINTER TO SPR FILE
        MOV C,M                 ;C-REG = LOW BYTE OF SPR CODE LENGTH
        INX H ! MOV B,M         ;B-REG = HI BYTE OF SPR CODE LENGTH
        PUSH B                  ;PUT CODE LENGTH ON STACK
        INR C ! DCR C           ;CHECK FOR EVEN PAGE LENGTH
        JZ REL2                 ;SKIP IF ALREADY EVEN PAGE LONG
        INR B                   ;ELSE MAKE IT NEXT EVEN PAGE
REL2:   MOV A,D ! SUB B         ;COMPUTE WHERE TO LOAD THE CODE
        MOV D,A ! MVI E,0       ;AS CURRENT LOW MEM MINUS CODE LENGTH
        LXI H,BUF+256           ;GET START OF RELOCATABLE CODE
        POP B ! PUSH B          ;GET REAL CODE LENGTH BACK
        PUSH D                  ;SAVE NEW CODE LOCATION POINTER
RELM:   MOV A,M ! STAX D        ;MOVE UP RELOCATABLE CODE
        INX H ! INX D           ;BYTE FOR BYTE WITHOUT
        DCX B ! MOV A,B         ;ANY RELOCATION AT THIS POINT
        ORA C ! JNZ RELM        ;LOOP ON SPR FILE CODE LENGTH
        POP D                   ;RE-FETCH PTR TO RELOCATABLE CODE
        LXI H,BUF+256           ;GET START OF BUFFER CODE BYTES
        POP B                   ;AND LENGTH OF RELOCATABLE CODE
        DAD B                   ;INDEX TO START OF BIT MAP

; AT THIS POINT HL POINT TO THE RELOCATION BIT MAP, DE POINT TO
; THE MOVED CODE IN MEMORY THAT MUST BE RELOCATED AND BC HAS A
; COUNT FOR THE NUMBER OF BYTES OF CODE THAT MUST BE RELOCATED.

        PUSH D                  ;PUT PAGE OFFSET ON STACK
        XCHG                    ;HL -> CODE   DE -> BIT MAP PTR
        PUSH B                  ;PUT BYTE COUNT ON TOP OF STACK

; FETCH THE NEXT RELOCATION BIT MAP BYTE AND DO 8-BYTES OF
; RELOCATION USING THAT MAP BYTE.  (OR UNTIL END OF RELOCATABLE
; CODE IS FOUND.)

RELNMAP:
        MVI C,8                 ;8 BITS PER MAP BYTE
        LDAX D ! MOV B,A        ;FETCH NEXT MAP BYTE -> B-REG
        INX D                   ;BUMP POINTER TO MAP BYTES
```

file with LINK.

XLOAD expects to find the name of the SPR extension file on the command line that starts it. To use XLOAD you enter a command like:

A>XLOAD MYBIOS <CR>

```
; SHIFT OUT AND TEST RELOCATION BITS.  IF A BIT IS A ZERO THEN
; THE CORRESPONDING BYTE DOES NOT HAVE TO BE RELOCATED TO RUN
; AT ITS NEW ADDRESS.  IF IT IS A ONE, THEN AN OFFSET FOR THE
; NEW BASE ADDRESS PAGE MUST BE ADDED TO THE BYTE.

RELNBYT:
        DCR C                   ;COUNT BITS IN MAP BYTE
        JM RELNMAP              ;GET ANOTHER MAP BYTE IF NEED IT
        MOV A,B ! RAL           ;FETCH CURRENT BIT PATTERN
        MOV B,A                 ;SHIFT OUT ANOTHER MAP BIT
        JNC NOREL               ;SKIP IF BIT IS A ZERO
        INX SP ! INX SP         ;OFFSET IS 2ND ON THE STACK
        POP PSW                 ;FETCH PAGE OFFS
        PUSH PSW                ;AND RE-SAVE I
        DCX SP ! DCX SP         ;READJUST THE STACK POINTER
        ADD M ! MOV M,A         ;ADD OFFSET TO MEMORY BYTE
NOREL:  INX H                   ;NEXT MEMORY LOCATION
        XTHL                    ;GET BYTE COUNT (TOP OF STACK)
        DCX H                   ;COUNT BYTE RELOCATED
        MOV A,H ! ORA L         ;SEE IF DONE
        XTHL                    ;PUT BACK COUNT / GET MEMORY PTR
        JNZ RELNBYT             ;CONTINUE IF MORE TO DO

; NOW WE GO AHEAD AND SAVE THE CONTENTS OF THE ORIGINAL BIOS JUMP
; TABLE FOR REFERENCE INSIDE THE NEW BIOS MODULE.  THEN WE MODIFY
; THE ORIGINAL BIOS JUMP TABLE SO IT GOES TO THE CORRESPONDING
; POINTS IN THE JUMP TABLE AT THE BEGINNING OF THE NEW MODULE.

        POP B                   ;DROP COUNTER FROM THE STACK
        POP H                   ;MAKE HL = NEW BIOS MODULE ADRS
        PUSH H                  ;AND SAVE ADDRESS BACK
        LXI B,0033H ! DAD B     ;GET TO PLACE TO SAVE OLD JMP TABLE
        LDA BOOT+2              ;FETCH BIOS BASE PAGE ADDRESS
        MOV D,A ! MVI E,0       ;THEN GET BIOS JMP TABLE POINTER
MOVBJ:  LDAX D ! MOV M,A        ;MOVE THE ORIGINAL BIOS JMP TABLE
        INX H ! INX D           ;INTO THE NEW BIOS EXTENSION
        DCR C ! JNZ MOVBJ       ;TO SAVE ORIGINAL ENTRY POINTS
        POP D ! PUSH D          ;NOW DE = NEW BIOS MODULE
        MVI E,3                 ;PLUS THREE
        LHLD BOOT+1             ;HL = NORMAL BIOS + 3 ADDRESS
        MVI C,16                ;NUMBER OF BIOS JUMPS TO FIX
NEWJMP: INX H ! MOV M,E         ;REPLACE BIOS JUMPS TO GO TO
        INX H ! MOV M,D         ;CORRESPONDING PLACES IN THE
        INX H                   ;NEW BIOS MODULE JUMP TABLE
        INX D ! INX D ! INX D   ;MAKE NEXT JUMP ENTRY ADDRESS
        DCR C ! JNZ NEWJMP      ;COUNT MODIFICATIONS AND LOOP
        POP D ! PUSH D          ;RE-FETCH ADDRESS OF NEW BIOS
        LHLD BDOS+1             ;REMEMBER OLD BDOS JUMP POINT
        PUSH H                  ;PUT IT ON TOP OF THE STACK
        LXI H,BDOS+1            ;GET TO LOW CORE BDOS JUMP
        MOV M,E ! INX H         ;AND MAKE IT GO TO OUR
        MOV M,D                 ;NEW BIOS MODULE
        POP B                   ;GET OLD BDOS ENTRY ADDRESS
        POP H
        INX H ! MOV M,C         ;PUT OLD BDOS ADDRESS INTO
        INX H ! MOV M,B         ;THE NEW MODULE CONTINUE JUMP
        EI                      ;RE-ENABLE INTERRUPTS
        RST 0                   ;DO CP/M WARM BOOT

; SEND AN IN-LINE CODED ASCII TEXT MESSAGE TO THE CONSOLE.
; THE TEXT CHARACTERS ARE TERMINATED WITH A NULL BYTE.
;
; FOR EXAMPLE:
;       CALL MSG
;       DB 'THIS IS THE MESSAGE',0

MSG     EQU $
        POP H                   ;GET NEXT CHARACTER ADDRESS
        MOV A,M ! INX H         ;FETCH CHARACTER AND NEXT
        PUSH H                  ;SAVE POSSIBLE RETURN ADDRESS
        ANA A ! RZ              ;RETURN AFTER ZERO BYTE
        MOV E,A                 ;ELSE CHARACTER -> E-REG
        MVI C,CONOUT            ;GET CONSOLE OUT FUNCTION CODE
        CALL BDOS               ;AND SEND THE CHARACTER
        JMP MSG                 ;CONTINUE

        DS 64                   ;ALLOCATE A 64-BYTE STACK
STACK   DS 2                    ;MAKE PLACE TO SAVE OLD <SP>

BUF     EQU $                   ;READ SPR FILE INTO HERE
```

It automatically supplies the file type "SPR" if you do not. If XLOAD can't find any file at all, or can't open the one you specify, it exits with an error message.

If the file is successfully opened, it is read into memory, moved up to the highest possible address in the TPA and relocated to run there. Once the new code is in place, the BIOS tables are adjusted and the location 0005H jump and the jump at the start of the new BIOS module are set for the new system.

When XLOAD has loaded an extension module, it exits via a warm boot, so you will see the warm boot message if all went well.

Some other thoughts

What happens if you have a BIOS extension at the top of the TPA and then load another extension? With the loader program shown the second extension will be loaded just below the first extension and its handlers will be used for the BIOS operations. You can try this by loading the same extension twice (or three or four times). Only the handler routines in the new extension will be recognized.

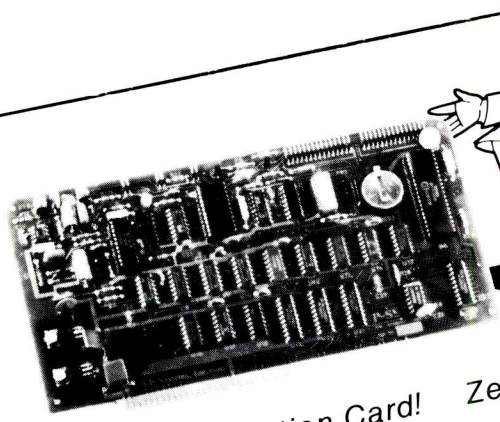
The loader program determines if it is running in a virgin CP/M system where the location 0005H jump goes right to the BDOS. If something is already in place below the CCP, the loader puts the BIOS extension immediately below that. If you try to use the BIOS modules described here along with some other CP/M extension (like a shell or whatever) they may or may not work correctly; so beware.

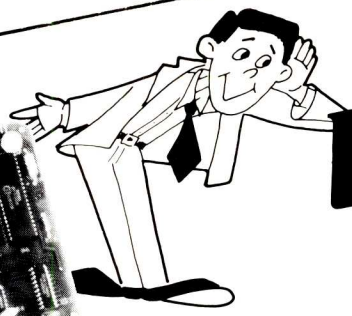
This whole scheme of code running at the top of the TPA can have other uses. For example, you can put an interrupt driven process there instead of a BIOS extension. It can run concurrently with other CP/M programs.

Since the BDOS entry at 0005H now goes to the base of your loaded module, it has access to all BDOS calls from CP/M programs. If you provide the necessary linkage into your module and then on to the BDOS, you can trap and modify the actions of the BDOS for special applications.

Summary

This article has described a way that you can modify the operation of one or more of the BIOS handlers in a CP/M system without changing the BIOS itself. A loader program is given that allows you to install these "loadable drivers" and hook them into the BIOS. You can have any number of BIOS routines for special applications and load and run them as they are needed.





PSST

Super Multifunction Card!

Zenith Z-100 • IEEE-696

- REAL-TIME CLOCK CALENDAR
  - Alarm, Heartbeat, Standby Interrupts
- THREE-VOICE SOUND/MUSIC SYNTHESIZER
- EXTENSIVE SOFTWARE SUPPORT
  - MS-DOS, ZDOS, CP/M
- TWO PARALLEL I/O PORTS
  - Joystick Compatible
- SPEECH SYNTHESIZER
- AUDIO OUTPUT
  - Pre-amp and Power amp

$395!

P-SST

The Programmable Speech/Sound/Time Card by LP Systems, Inc. is a multi-function peripheral card designed to IEEE-696 (S-100) bus standards. The best selling accessory card for the H/Z-100 series of computers from Heath Co. and Zenith Data Systems, it is also adaptable, with appropriate software, to most systems which meet the IEEE-696 standard.

The **P-SST** combines those functions and features most requested by H/Z-100 users and programmers. Complete schematics and documentation, as well as extensive software support, make the P-SST useful to the novice as well as the advanced programmer. The P-SST is an "open" system, and user development of applications software is encouraged.

SOFTWARE

Current distribution and development software supports the use of the H/Z-100 series of computers under MS-DOS/Z-DOS, and CP/M-85/86 and any CP/M 2.2 IEEE-696 system.

Distribution software for the H/Z-100 includes automatic demonstration batch files, and demonstration programs for speech, music, graphics, sound, and joystick use.

Utility programs include a hardware diagnostic program for the P-SST, background music interrupt, and a clock read program to automatically set the time and date in MS-DOS/Z-DOS on boot-up.

Three-voice ASCII music scores are provided for use with the demonstration programs, and as examples of how to prepare scores for input to the MUSIC program. Notation used is similar to that for the IBM-PC BASICA "PLAY" command, for which many scores are available.

The SPEECH program will output words through the speech synthesizer from a specified ASCII text file.

Optional software includes the **P-SST Development Libraries** ($49.95), for use with most high level language compilers, and the **MAESTRO Music Editor** ($59.95) for music scoring and composition. The Development Libraries include over 50 subroutines, functions, and source code for most of the distribution demonstration and utility programs. MAESTRO allows graphic display and editing of music scores, background music utility programs, and an additional 50 three-voice music scores.

P-SST is also supported by the following **Software Wizardry** programs:

CHRONOLOGIC clock program (clock/calendar)

PALETTE color graphics editor (joystick)

REACTOR-100 Nuclear reactor simulation (sound/voice)

ZLYNK/II smart modem communications (clock/calendar)

ESP Bulletin Board/Dial in (clock/calendar)

Software Wizardry
THE MAGIC TOUCH

Dealer Inquiries Invited

1106 First Capitol
St. Charles, MO 63301
(314) 946-1968

ZENITH data systems

Firepower!

Winchester Systems for the Zenith Z-150

- 11, 20, & 36 Megabytes internal
- Includes controller, drive, cables and instructions
- Specially engineered for the Z-150 by Software Wizardry
- No software mods-boot from hard disk
- 11 Meg drive system $ 895
- 20 Meg drive system $1195
- 36 Meg drive system $1595

System Specials!

Z-150 with:
640K RAM $3495.00
1 floppy
36 Meg hard disk
Green or Amber Zenith Monitor

Same system, but with
20 Meg hard disk
$2995.00

Call for Complete Catalog!

1-800-TO-BUY-IT (orders only)
1-314-946-1968 (other info)

First Capitol Computer™
1106 First Capitol Drive
St. Charles, MO 63301

Please add 2% for shipping in USA

First Capitol Computer

The UNIX File

by Ian F. Darwin

The UNIX File looks at many aspects of the UNIX operating system. If you have comments or questions about UNIX or this column, feel free to write to Ian Darwin at Box 603, Station F, Toronto, Ontario, Canada M4Y 2L8.

If you have UNIX mail access to the UUCP network, you can contact me at "ihnp4!darwin!ian". I can't always answer immediately, but I will get back to you (electronic mail gets answered first!). And I'm always glad to hear from readers with comments either on the column itself or on their reactions to particular UNIX systems or products.

It's good to be "back'. Regular readers of Microsystems magazine will recall this column's previous incarnation there. For new readers, I'll just say that I try to offer an eclectic mix of useful information on and about UNIX and happenings in the known UNIXverse. As I said in the very first UNIX File, and it's still true two years later, UNIX is being seen, spoken about and advertised today by people who don't know that much about it. The UNIX File is a new column that will focus on selected aspects of UNIX.

My aim is to dispell myths, provide useful information, spotlight new products (or interesting older ones), and occasionally steer people clear of lemons. In this issue I describe the Dallas conference, look at the new *Bell Labs Technical Journal* on UNIX, discuss IBM mainframe UNIX, and conclude with some short notes.

Dallas in January

The Crystal Palace transplanted to Texas? Companies *giving away* UNIX systems? Reading USENET news bounced off a satellite? An inter-hotel shuttle bus system that worked? You could have seen all these improbabilities by joining the annual UNIX gathering of the clans in Dallas in January, 1985.

The Crystal Palace, built in London in the last century for the first World's Fair, had an architecture far ahead of its time and has been copied in several parts of the world. Among the more ambitious copies is the Dallas Infomart, a permanent display for many computer companies and a large exhibit hall for shows such as UniForum. The January '85 /usr/group trade show was also the first show in the new Infomart, and UniForum or USENIX attendees could attend the grand opening. The new center offered plentiful space for the large gathering of UNIX vendors. A list of who was there would fill my entire column, so 'nuff said. If they sell in the UNIX market, they were probably there.

And yes, there were at least two companies giving away UNIX boxes. I didn't win either of the two new HP UNIX portables nor the two GOULD workstations, but four people now have UNIX on their desk as a result of entering these draws.

The two winners of the HP portable can have UNIX in the field too! Not a lap computer but a full-scale portable UNIbox, the HP has a dot-addressible screen usable either as a conventional terminal or as a window system with a mouse. UNIX wins over many 8-bit window systems, since it's a true multiprogramming system and you can have several processes each writing to its window concurrently. And the HP system includes a full UNIX system, with the kernel cleverly in PROM. Of course it is copied into RAM when it boots up, so that it can be patched (or customized). At under $5K US, it's affordable for the executive wanting a portable floppy-based UNIX with an optional hard disk that stays at the office. I wish I'd won one!

Hardworking Lauren Weinstein was there, despite some last-minute uncertainties, demonstrating an experimental method of distributing USENET news — via satellite. The *experimental* (so far!) link uses the video retrace interval on a domestic television satellite to transmit news, in a fashion similar to closed-caption coverage or stock-market services or Telidon/NAPLPS service. Many hurdles remain before it could be used to cut down on production USENET's phone bills, but the demonstration went smoothly and as planned. Kudos to Lauren for making it happen, and to the USENIX association for backing the project.

And there were two full days of technical talks on UNIX. Conference co-host Rob Kolstad gave an amusing keynote talk on 'gurus'. A wide range of topics was covered, and your columnist gave a talk on UNIX Programming Style. Proceedings are $20 (plus $15 for overseas airmail) from USENIX (see below). Kolstad and co-host Charisse Castagnoli as well as the program committee deserve a round of applause for organising an above-average technical program. And for organising evening trips, including the one to play Photon... The Fairmont Hotel was not inexpensive, but well situated and functional — even a twenty-four-hour coffee shop!

This marked the last "joint /usr/group-USENIX conference" for a while. Summer's USENIX technical conference is slated for Portland, Oregon in June 11-14, 1985, and will have both technical sessions and a trade show. Next winter's USENIX and UniForum will be held in different cities a month or so apart. It was fun for attendees while it lasted, but I guess the associations involved had too many differing priorities. They also had the problem of finding conference facilities big enough to hold both shows at the same time.

For information on upcoming USENIX conferences, write Box 7, El Cerrito CA 94530, or phone 415-528-8649, or mail ucbvax!usenix!office. Details on the 1986 UniForum can be had from /usr/group, 4655 Old Ironsides Drive, Suite 200, Santa Clara CA 95050.

Other upcoming conferences include the European UNIX User Group (April in Paris), C/85, billed as the 'First National Conference on C' (San Francisco, May 13-85, contact C/85 c/o Lifeboat, 1651 Third Ave, N Y, NY 10128), and UNIX EXPO, a trade show, in New York, September 18-20, 1985 (contact National Expositions, 14 W 40th St, New York NY 10018, 212-391-9111).

Let's Get Technical...

Wanna get technical? Apart from wading through the source code, a main source of UNIX information has been the *BSTJ* issue on UNIX. As I wrote in the November issue of *Microsystems*, "After the manuals, another important series of papers... appeared in the *Bell System Technical Journal*, July-August 1978. This special issue [is] part 2 of the July-August 1978 issue... The magazine is now called *Bell Laboratories Technical Journal* and is doing another special issue on UNIX that should come into print around the same time as this article. Watch for it!"

That issue came into print as and when predicted, and is now being distributed. The list of contributors reads like a who's who of current UNIX research and development inside AT&T and Bell Labs. The table of contents (see box) provides an idea of the breadth of topics covered. If you're interested in the technical side of UNIX — what's inside, new user and system interfaces, history, etc. — get a copy. But it's not cheap — the price is $27.50, and it has more typographical errors than you'd hope for. Order from ATT Customer Information Centre, Box 19901, Indianapolis IN 46219.

Table of Contents — Bell Labs Technical Journal, October 1984

Preface: R. L. Martin
Foreword: A. V. Aho
The Evolution of the UNIX Time-sharing System: D. M. Ritchie
Program Design in the UNIX System Environment: R. Pike, B. W. Kernighan
The Blit — A Multiplexed Graphics Terminal: R. Pike
Debugging C Programs With the Blit: T. Cargill
UNIX Operating System Security: F. T. Grampp, R. H. Morris
File Security and the UNIX System Crypt Command: J. A. Reeds, P. J. Weinberger
The Evolution of C: L. Rosler
Data Abstraction in C: B. Stroustrup
Multiprocessor UNIX Systems: S. J. Buroff, M. J. Bach
A UNIX System Implementation for System/370: W. A. Felton, G. L. Miller, J. M. Milner
UNIX Operating System Porting Experiences: D. E. Bodenstab, et. al.
The Evolution of UNIX System Performance: J. Feder
Cheap Dynamic Instruction Counting: P. J. Weinberger
Theory and Practice in the Construction of a Working Sort Routine: J. P. Linderman
The Fair Share Scheduler: G. J. Henry
The Virtual Protocol Machine: M. J. Fitton, C. J. Harkness, et. al.
A Network of Computers Running the UNIX System: T. E. Fritz, J. E. Hefner, T.M. Raleigh
A Stream Input/Output System: D. M. Ritchie

UNIX Goes Virtual

In February, 1985 IBM announced a System V port of UNIX for its mainframe computer family. This put to rest the silly and unfounded rumors that IBM was trying to drive a wedge between System III and ATT's newer "Consider it Standard" System V. (If they'd been in a wedgy mood, they could have gone with 4.2BSD). The big blue behemoth probably went with System III for its smaller PC line because a System III for the 8086 was conveniently available when they needed it. Look for the PC port to be upgraded to System V sometime soon. Details on the IBM mainframe port 'IX/370' would at least fill this column, but here's a brief summary. This UNIX runs as a timesharing system under VM/SP on supported IBM mainframes, offers full System V with **uucp** and **vi** and full-duplex ASCII terminal support (via channel-attached Series/1 front end), no 3270-series support, its own screen editor, and is otherwise a full and 'standard' System V port. Exceptions: no games (this is IBM, after all), and no on-line documents (does this mean no *man* command? I think not, but you never know...).

Giving big blue a run for its money is *Amdahl*. They announced their System V port for IBM (and Amdahl) mainframes at UniForum, a few weeks before IBM. Amdahl has had a primitive UNIX port out for several years; their new product, also a 'Standard System V', needs an Amdahl 4705 front end but supports both full-duplex ASCII and IBM 3270 terminals. These are the third and second public IBM ports; the first is from ATT but is not widely available outside the labs. See the BLTJ issue mentioned above for a paper on the ATT port; contact your IBM and Amdahl rep for details on their products.

What do these two ports mean? Mainframe UNIX is more widely available than ever. And with the support of two megacorporations, both of them pushing ATT's System V standardization effort, UNIX runs on more computers from micros to mainframes. The effects on UNIX direction and evolution of having IBM among the main players, however, may not be what UNIX old-timers had in mind. Only time will tell.

Miscellany

More UNIX Publications: First is *The Business UNIX Journal*, whose introduction says: "... tens of thousands of Unix, Xenix and compatible systems are already in place and hard at work providing business and office automation functions. Many more systems are bound to be added to that figure. Our charter... is to provide a communication forum and an information service to the users and the vendors of these systems." Write them at Box 1065, Lafayette, CA 94549. Phone 415-284-1610. The price is $28/year for six issues. And not one two journals on the C language. *The C Journal* is from InfoPro, at 3108 Route 10, Box 849, Denville NJ 07834, phone 201-989-0570, UNIX mail harpo!infopro!cj. It is "aimed at the technical people directly involved in using C and at [their] managers..." Cost is $28/year (4 issues), plus $9 for overseas airmail. And */C* from Cue Publications, 7999 Knue Road, Indianapolis IN 46250, phone 800-227-7999 or 317-842-7162. Price is $60/year for 12 issues. *And software coming over the phone...* AT&T is pioneering a new method of selling software. You not only order by phone, but they deliver it by phone (after all, this is the phone company, right?). You order by dialing into a menu system, and the software is delivered by *uucp*. Called the "Software Toolchest", it is filled with software that is usable but for one reason or another did not become formal products. Prices I saw ranged from $40 (the source to a video game) to several thousands of dollars for *ksh* , a UNIX shell interpreter that has been much discussed of late. Also listed were editors, languages, debuggers, etc. All this software is tested, but not supported; the database tells you what machines and systems it has run on. The Toolchest is being opened up in stages; you must have a System V *source* license to use the Software Toolchest during phase one. Phase two, slated for later in '85, will make it available to anybody with an interest in UNIX software. The data phone number is 201-522-6900, login as 'guest', voice phone is 201-522-6698 (answered 9-5 EST).

That's all for this month. Cards and letters and electronic mail welcome on these and other topics, especially suggestions for future columns. Cheers!

The CP/M Bus

by Bruce R. Ratoff

The Public Domain Hit Parade

One need only work with CP/M for a relatively short while to know that it is a solid, reliable system in most respects. One of its unfortunate shortcomings, however, is that only the most rudimentary of software tools are provided with the standard system. Most CP/M programmers soon begin to accumulate additional utility programs, creating their own "bag of tricks" from which they can draw to make their work easier. Luckily for all of us, many top-notch programmers have generously decided to share the tools they have created with the rest of us, by placing them into one of the public-domain software libraries, such as SIG/M. Some of these programs have become so popular that many of us would feel lost without them. Some programs seem to actually develop a life of their own. While the original authors watch with a sometimes Frankensteinian awe, version after version, containing different programmers' improvements, continue to appear in the public domain libraries and Remote CP/M bulletin boards across the U.S. We can hope that the original author will view this phenomenon as a compliment to his/her efforts, and continue to place other useful tools in the public domain.

What sort of tools are we talking about? Well, although everyone has their own preference, based on their own tastes and work habits, I thought it might be useful to talk about some of the public-domain programs that I wouldn't want to be caught without. This is by no means an exhaustive list. It was derived simply by skimming through the directories of a few of my working disks. This listing therefore reflects those programs which I find useful in my own day-to-day programming activities.

Compare

This simple little utility compares one binary file with another, byte by byte, and displays the differences in hex. It is useful for deciding whether two .COM files with the same name are really the same file. More importantly, you can discover if the differences are minor (a last minute patch, for example) or major. You won't trot this one out every day, but there are times when nothing else will do.

CRC and CRCK

Two different variations on the same basic idea: compute a unique, 16-bit "magic number" from the arbitrary contents of a file. The technique used in computing a "CRC" (Cyclic Redundancy Check) value treats the file as a bit stream, making it resistant to being fooled by bit position, byte position or special values. Both programs handle wildcards, and will optionally generate a file containing all the filenames and their CRC values. In addition, CRC.COM will later read a file containing filenames and CRC's, recompute the CRC's of all files, and report any differences.

There are many uses for this program. It will quickly tell you if two files are identical, by whether they have the same CRC value (admittedly, there is one chance in 65536 of being wrong!). It is similarly useful after modeming a file to someone else, as a final check that the file arrived intact. If you periodically generate a CRC list for all the files on a disk, you may then use CRC later on to see what files have been changed. Finally, when making backups of a set of files from hard disk to floppy, I put a CRC list on the floppy as an extra means of checking my backup copies. I'm sure you'll come up with additional uses for this program.

DIF2 and SSED2

DIF2 does a line-by-line ASCII compare of two files. Its intended use is to list all the changes between two different versions of a program in source code form. This program is written in the C language, and is actually an adaptation of a UNIX system utility, but it will work on source programs in just about any language. UNIX-style input and output redirection is supported, so you can generate a file of all the differences. There is also an option to output the differences in the form of a series of editing commands. The resulting file of edit commands, when fed into the SSED2 program along with one version of the program, will produce the other version of the program. This "difference file" technique provides a compact way of distributing source code updates. It is frequently used on the bulletin boards to avoid having to download an entire source file just to pick up a few minor changes.

DU

If I had to pick one program that I wouldn't want to be without, this is it. DU stands for "Disk Utility", and what it allows you to do is perform "brain surgery" on your CP/M disks. Using DU, you can easily examine and modify any byte on a disk. You can search for a particular string of bytes anywhere on the disk, position anywhere on the disk by track and sector or by CP/M block number, copy sectors from one place to another and fill sectors with a particular value. You can also get a usage map of a disk, showing what CP/M blocks belong to which files. There is a simple command macro facility to make repetitive jobs a bit easier. If you know the internal layout of a CP/M directory, you can use this program to unerase files, reconstruct lost backups, etc. I also use it occasionally to make simple patches (it's a whole lot quicker when your changes are small).

FIND

This program will search through a file or group of files (it takes the normal * and ? wildcards) and display all lines containing a particular string. I find it very useful while working on a large program to find all the references to a particular label or variable. Unlike the search command in most editors, FIND will locate all occurrences of the string in both upper and lower case with a single command. Not having to use an editor to perform this function has saved me hours and hours.

LU

LU stands for "Library Utility", and what it allows you to do is to take a bunch of related files or programs and stick them together into one much larger file. What you are then left with is a single file that can be copied or modemed around, and later broken back into separate files again using LU. This is a popular utility on the bulletin boards for "packaging" all the files associated with a program. If you have a lot of small files, it can also be a space saver, since space in the library file is allocated in individual sectors rather than CP/M blocks. LU itself contains commands for adding files to a library, extracting them, deleting them, taking a directory of the library, and reorganizing a library to reclaim wasted space from deleted files. There are additional separate utilities for displaying, modeming and running files from a library without extracting them first.

(continued on page 77)

Micro/Systems Journal Back Issues

The first issue of *Micro/Systems Journal* is still available. Send $4 per copy ($5.50 foreign, cash or U.S. Bank check) including shipping.

Microsystems Back Issues

The following back issues of the old Microsystems magazine are still available. Quantities of many of the issues are, however, limited. They are $3.50 per copy ($5.00 foreign, cash or U.S. bank check) including shipping. If ordering 3-9 copies deduct 10%, 10 or more copies deduct 15%. Make check out to "Micro/Systems Journal", Box 1192, Mountainside NJ 07092.

1984

NOVEMBER: Unix Development; MS-DOS Prompt Command Features; X.25 Communications Protocol - Part 3; How Portable is C?; S-100 Power Switching and dealing with slow Eproms; Unix MAKE utility and a shell "exec"; Bulletin Boards for the PC; REVIEWS: PFIX-86, Codata 3300, HALO.

AUGUST: Intro to Local Area Networking; Graphics Subroutines in C For NAPLPS; Using YACC, MAKE and Prolog under Unix; Multiprocessing on S-100; Using Unix Sort, ciphers and enhancements; REVIEWS: TurboDOS, NCR-PC, MindSet-PC, Adding TurboDOS to NorthStar System, Leverage DBMS for Unix.

JULY: Intro to Computer Graphics; Intro to NAPLPS; NAPLPS Directory; Graphics on DEC PRO/350, NCR-PC and Mindset; Speed up S-100 front panels; Unix Portability; REVIEWS: DRI-GSX, Princeton Graphics HX-12, Quickpel, Createx, Videophile, DRI-PL/1-86, Mitsubishi half-height 8" drives.

JUNE: Implement X.25 Communications Protocol, RCPM and RPC Systems; RCPM Directory; Computer-to-Computer File Transfers; Log onto your system as a Remote Terminal; 8250 UART Interfacing; S-100 Interrupts, clock signals, RTC and power requirments; REVIEWS: ASCOM, 212A modems.

APRIL: Unix Software Directory; Upgrade NorthStar ZPB; MS-DOS 2.0 Overview - Part 2; S-100 Phantom & Bank Selecting; Upgrading FIG Forth; REVIEWS: UniPlus+, Informix, DRI-C.

FEBRUARY: Using WordStar to Create Mailmerge/DBase-II files; Moving data files between CP/M software packages; Datestamp DBase-II; CP/M 2.2 Deblocking; Building S-100 diagnostic hardware; Enhance CP/M+ with RSX; REVIEWS: DBase-II, S-100 Mainframes, DRI Display Manager, AutoDex, Turbo Pascal.

JANUARY: Enhancing MP/M - Part 1; Installing MP/M; Add Concurrency to MP/M; Two Users on CP/M; Relocating Assemblers & Linkage Editors - Part 3; S-100 Wait States; REVIEWS: MP/M-8/16, ProComp-8, Paragraphics Game Board, ProLog.

1983

DECEMBER: CP/M Software Directory; A Debug Subroutine; Implement IOBYTE on North Star; Floppy Disk Problems; Improve Trig Functions in CBasic-80; Build Cheap S-100 Memory; Extended Memory Management; CP/M-86 BDOS Calls; REVIEWS: XLISP, LISP/80, TLC LISP, APC Basic, Microdynamics S-100 EProm Programmer, Ackerman S-100 Digital Synthetalker, Digital Research 16K & 32K S-100 Memory cards.