For the Advanced Computer User

Micro/Systems Journal™

BRINGING UP CP/M-PLUS

see pages 20-28

Also in this Issue

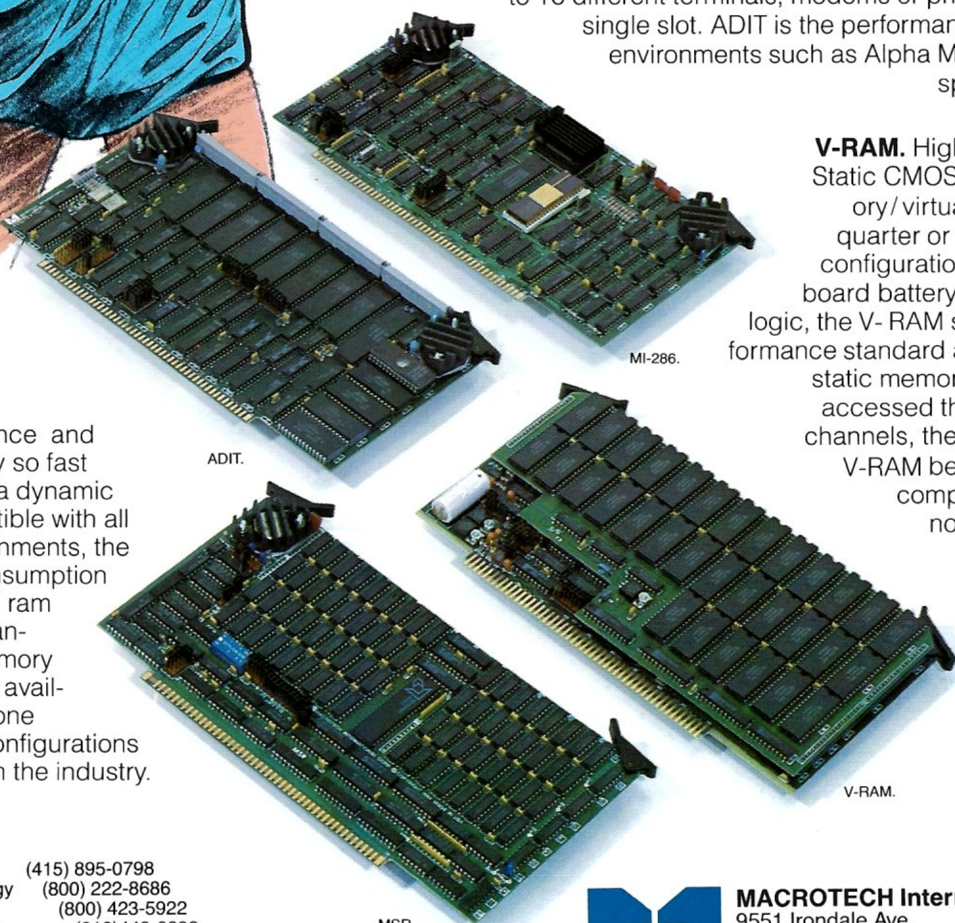
Complete Table of Contents on Page 3

March/April 1985 Vol. 1/No. 1 $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.

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.

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

IBM-XT/AT™ AND S-100 SYSTEM INTEGRATORS...

Be a Leader...

in the emerging AutoCAD™ market

FOR DETAILS CALL: (800) 345 MOMS (CA)
(800) 255 ACAD (Outside CA)

EUCLID

EUCLID SYSTEMS, INC.

76 Belvedere St., Suite D, San Rafael, CA 94901

Suppliers to CAD Dealers and Systems Integrators

Teleteknology

One Success After Another.

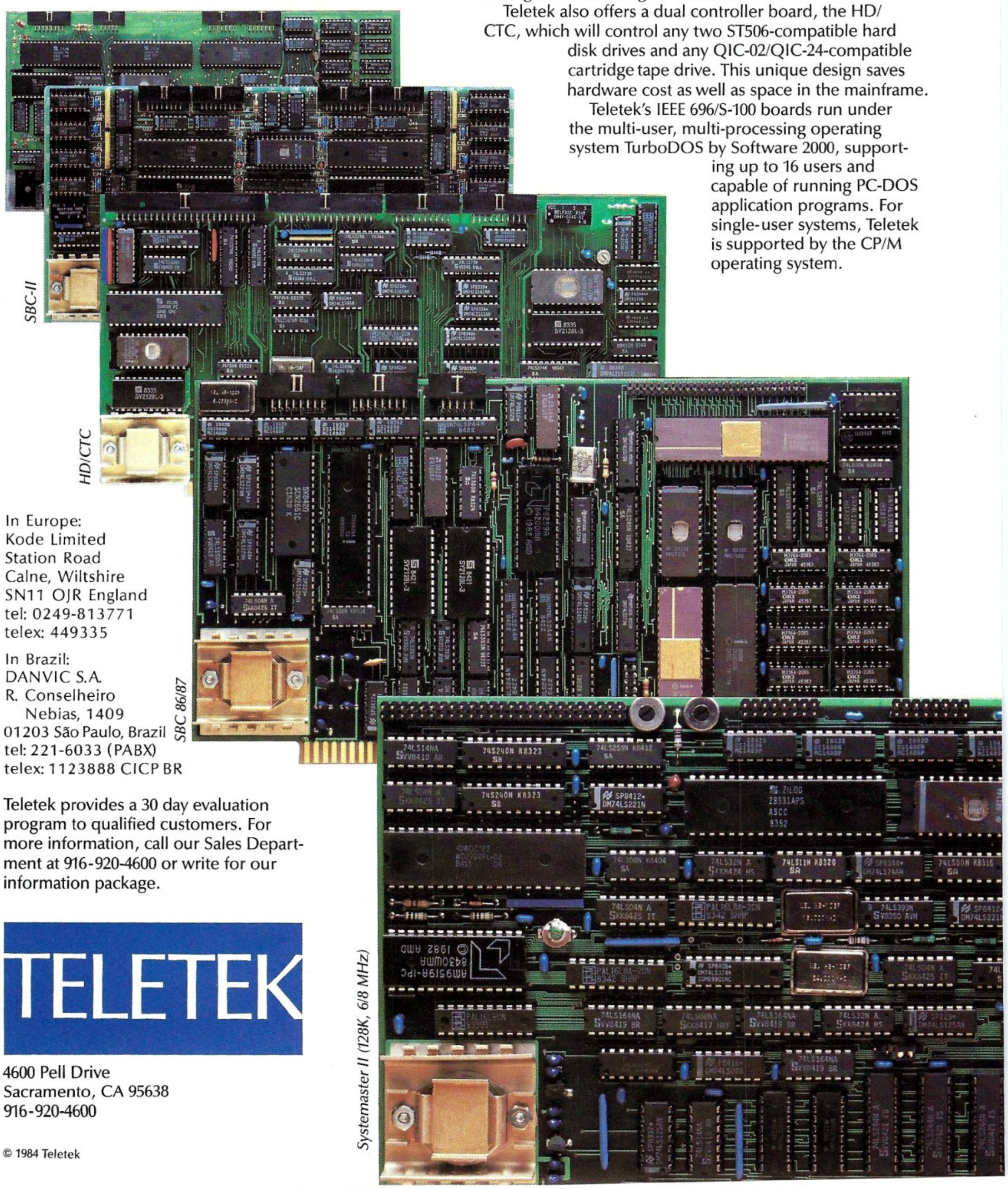
Since 1968 Teletek has been a leader in the design and manufacture of single board computers, controllers, memory boards and interface boards.

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™

March/April 1985
Vol. 1 No. 1

SUBSCRIPTION RATES	Bulk Rate	First Class
1 year	$18	$24
2 years	$32	$44
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.

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 REVIEWS

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.

Editor's Page

Way back in 1979 I came to the conclusion that there was a need for a magazine that catered to CP/M and S-100 systems users. I spent several months trying to convince one publisher after another to publish the magazine, with no success. Thus, by the end of '79 I came to the realization that the only way my brainstorm would see the light of day was for me to do it myself. The first issue of "Microsystems" magazine, my baby, appeared in January of 1980.

I had intended for Microsystems to be just a small little magazine with a few hundred subscribers and to provide a communications medium for CP/M and S-100 users. My wife Lennie and I were ill-prepared for the huge response and the incredible amount of work involved in publishing a magazine. There is a lot more than just writing and editing, believe me! Thus, after several months of working seven days a week, from dawn into the wee hours, we decided to get out of the publishing business while retaining the editorial end of the magazine.

Hence, in mid 1980 we sold Microsystems to Creative Computing who published it during the later part of 1980, all of 1981, and into the beginning of 1982. It was then that Ziff-Davis bought Creative Computing and acquired Microsystems as part of the deal.

Ziff-Davis is a very large magazine publisher with many magazines and several hundred employees. Microsystems had one of the smallest circulations of all their magazines. Z-D considered Microsystems to be a very prestigious publication and therefore decided to continue publishing it and in fact invested quite heavily in trying to increase the circulation and advertising. However, after two years Z-D decided to call it a day and reluctantly closed it down.

Looking to the future

A great deal has changed in the 5+ years since I originally started Microsystems. Thus, it is only logical to expect that MICRO/SYSTEMS JOURNAL is going to be different from its predecessor. Although S-100 systems are still viable their numbers now account for only a small fraction of the systems in use. Their strength today lies in the multi-user, multi-processing and high performance systems areas. MICRO/SYSTEMS JOURNAL will continue to provide support for S-100 users as it applies to these higher performance applications.

CP/M is today largely the domain of low cost single-user systems such as the KayPro, Commodore-64 and Apple-IIe with Z80 plug-in cards. In fact there are probably more Apple-based CP/M systems in operation then all the other types of CP/M systems put together. And the second most popular CP/M system is probably the Commodore C-64. KayPro probably ranks a distant third.

The more sophisticated users are moving to MS-DOS for single user applications and Turbo-Dos and Unix for multi-user, multi-processing applications. Xenix appears to be the current leader here (with Tandy and Altos machines) and the introduction of Xenix on the IBM-PC will no doubt assure it the dominant position for small micro-based multi-user Unix systems. In fact the four largest suppliers of S-100 systems are using Unix and many others are using Turbo-Dos. Thus, we will also provide support for systems such as Unix, Turbo-Dos, CP/M and other high-performance operating systems.

Several S-100 manufacturers have already introduced systems running MS-DOS with several more planning the same. Further, a number of S-100 makers are introducing multi-user concurrent CP/M-86 systems with MS-DOS compatibility. Therefore we will support 16-bit operating systems such as MS-DOS and CCP/M.

I will be giving much more coverage to the PC and CP/M public domain software areas then was previously given in Microsystems. I believe that public domain software is one of the most exciting parts of the microcomputer scene.

Further, the PC-bus has become a defacto hardware standard as many manufacturers make systems and plug-in cards using this bus. In fact, the PC-bus has a much wider acceptance than the S-100 bus. We intend to run articles on interfacing to the PC-bus. For example, in our next issue you will find an article on building an S-100 to PC-bus interface. And, starting in the next issue Dave Hardy (whose S-100 Bus Column ran in the old Microsystems) will begin a regular column on PC-bus hardware interfacing. If you have an IBM-PC or compatible and want to improve its performance, interface some unusual devices to it or are having compatability problems, send in your letters and Dave will attempt to answer them.

What a time to start a magazine!

During the last 8 months close to two dozen computer magazines and computer book publishers have folded and several others are just barely hanging in. The future does not seem to hold great promise for the computer magazine publishing industry.

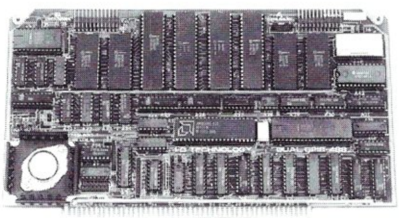
Further, there is no doubt that a magazine that caters to the very sophisticated microcomputer user will have a limited circulation and a limited advertiser base. We realize that it will never really be published by a large

Continued on Page 13

INPUT/OUTPUT TECHNOLOGY, INC.

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

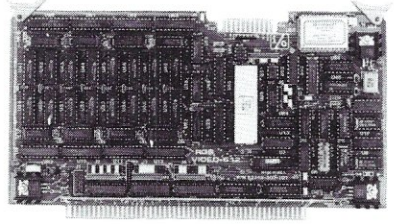
Uncompromising Additions to your S-100/IEEE-696 BUS



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.

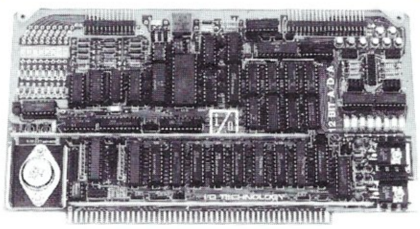
A&T, P/N 52748-800-102



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 ...

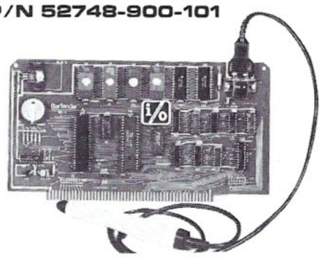
A&T, P/N 52748-300-101



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

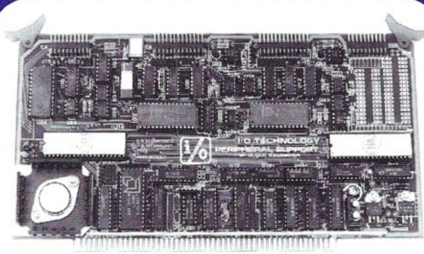
A&T, P/N 52748-900-101



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.

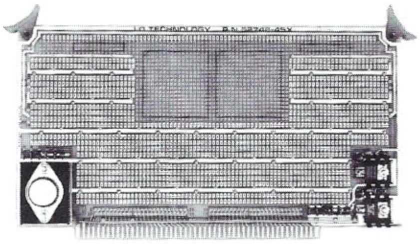
A&T, 52748-500-101 Without Wand
A&T, 52748-500-201 With Wand



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



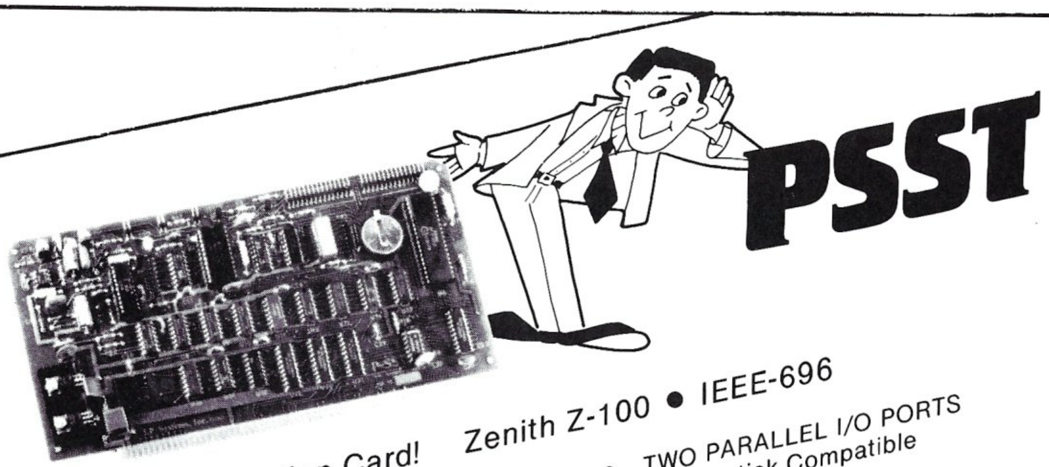
MULTI-PURPOSE PROTOTYPING KIT

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

SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE.



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

IBM is a trademark of IBM, Inc.
Heath and H-100 are trademarks of Heath Co.
Zenith, Z-100, Z-DOS, and Z-BASIC are trademarks of Zenith Data Systems.
MS-DOS is a trademark of Microsoft, Inc.
CP/M is a trademark of Digital Research, Inc.
P-SST is copyright 1984 by LP Systems, Inc.
P-SST Software, MAESTRO, PALETTE, REACTOR-100, ZLYNK/II, and ESP are copyright by Software Wizardry, Inc. and the respective authors.

Dealer Inquiries Invited

1106 First Capitol
St. Charles, MO 63301
(314) 946-1968

ZENITH data systems

Firepower!

Winchester Systems for the Zenith Z-150

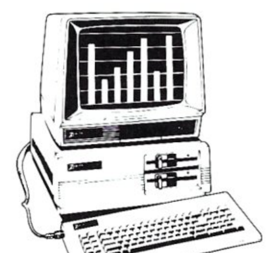
- 11, 20, & 36 Megabytes internal
- Includes controller, drive, cables and instructions
- Specially engineered for the Z-150 by Software Wizardry
- No software mods-boot from hard disk
- 11 Meg drive system $ 895
- 20 Meg drive system $1195
- 36 Meg drive system $1595

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

News, Views & Gossip

by Sol Libes

Gossip

Digital Research is reported developing an 80286 version of Concurrent PC DOS for release at NCC... AT&T is expected to shortly release Version 5.3 of UNIX System V which should have support for virtual memory, and file and record locking, features needed for success in the business marketplace... Multi Solutions, Inc. has signed its first OEM contract for its S1 operating system with a Japanese supermicro maker (Computer Engineering U& Consuting Ltd.)... Predictions are that IBM will ship 200,000 PC AT machines this year, up from an estimated 40,000 last year... It's rumored that 3M will shortly introduce an erasable 500Mbyte laser disk. It is not expected to be compatible with the read-only compact disk units beginning to appear from Japan.

Public domain software

Elsewhere in this issue you will find news columns specifically on SIG/M and PC/BLUE. However, there are some other organizations also distributing Public Domain Software.

The Houston Area League of PC Users (HAL-PC) furnishes disks from its library for $2/disk. For a listing of titles send a stamped self-addressed envelope to: Nelson Ford. HAL-PC librarian, Box 61565, Houston TX 77208. Clubs wishing to trade PDS software should contact: Jack McClure, Box 610001, Houston TX 77208.

Public Domain SW, 1400 Coleman Ave., C-18, Santa Clara CA 95050 (no phone given) has a library of public domain software for the IBM-PC and compatibles. There are 108 disks in the library and disk number 0 contains a catalog and ordering information. They charge $7/disk plus shipping. The information disk alone costs $8.65. I was not able to determine whether this operation was run by a club or is a private business.

The Public Domain Exchange, 673 Hermitage Lane, San Jose CA, (408)942-0309 is a private business providing support for Apple users including CP/M and the Macintosh. Their catalog of PDS software for CP/M on the Apple includes 91 volumes and they have 30 volumes of C-User Group PDS disks. They charge $15 for a catalog and $10/disk plus shipping.

The C Users' Group, Box 97, 415 E Euclid McPherson KS 67460, has issued three news volumes. They now have 45 Volumes available in 8"SSSD, IBM-PC, Osborne SSDD, Apple, Heath and North Star formats. 8" is $8/vol while 5.25" are $12/Vol.

The PC-SIG Software Library, 1556 Halford Ave., Suite #130, Santa Clara CA 95051, (408)730-9291, a private business, has a library of 222 disks for the IBM-PC and compatibles. A printed 2-volume catalog is $9. They charge $6/disk plus shipping.

Club news

The CompuPro Users Group has a bulletin board up and running at (703)491-1852. The group also has an RCPM system and publishes a newsletter for members. Membership is $20/yr. Call Don Kelley (703)690-3312, or write Toni Bennett, C-PRO UG, 14057 Jefferson Davis Highway, Box 1474, Woodbridge VA 22193.

DRI out/Microsoft in on UNIX

Digital Research has given up on its UNIX-V port project for AT&T. Several individuals from the DRI project have reportedly spun off into a new startup company in attempt to finish the project. DRI had a development contract with AT&T for the software and was known to be behind its scheduled delivery dates to AT&T.

In the meantime AT&T has turned to Microsoft and will do a certification evaluation of XENIX, Microsoft's version of UNIX for 68000, 8086 and 80286-based machines. Although the current version of XENIX is System-III compatible, Microsoft is promising a System-V version shortly.

So, it looks like Microsoft has done it again. DRI, who was once the leading supplier of single-user DOS software, is now a minor player in that area. And now, DRIs has given up on its attempt at being the leading player in the multi-user micro DOS marketplace.

DRI's appears to be betting on its new Macintosh-like operating system for ATARI (actually CP/M-68K with a GEM front end). If the machine does not succeed, what will DRI try next? It is interesting to note that DRI had contracts with Mattel, Timex, Coleco and other comsumer-oriented companies who are no more.

DRI & CP/M

John Rowley, President of Digital Research has reported that sales for fiscal year 1984 doubled to $74 million, compared to $37 million in '83. Further, he reported that CP/M revenue was about $9 million while revenue from Concurrent PC DOS was about $18 million. He indicated that the primary source of CP/M revenue was from Europe and Japan.

DRI is putting a strong emphasis on Concurrent DOS for the 80286 expecting it to be the dominant revenue producer in the future. It thus appears that, as far as DRI is concerned, CP/M's life cycle is coming to an end.

IBM update

Venture Development Corp., a Wellesley MA market research outfit, recently counted 3,000 separate commercial programs for the IBM-PC; 10% were accounting programs, 6% word processing, 5% database managers, and 3.6% spreadsheets. By contrast Apple claims there are about 5,000 Apple programs, however the largest portion of these are games.

By the time this column appears in print IBM's new TopView windowing user-interface software should be out. Although limited in capability compared to Digital Research's GEM and Microsoft Windows (which is not expected until June, at the earliest), it is

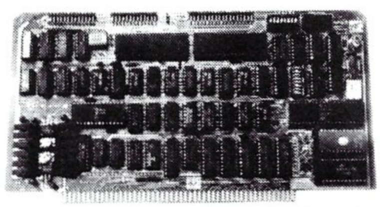
Quality S-100 Products

FULCRUM: *The S-100 Specialists*

Introducing...

FULCRUM'S NEW

MPUZ CPU



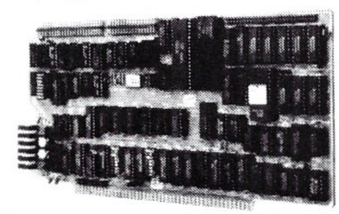
This NEW MPUZ CPU utilizes the Z-80 8MHz uP as a basis for its 8MHz CPU for S-100 systems, and has been carefully designed to meet the requirements of the IEEE - 696 standard. The quality and performance this CPU provides is rarely found in S-100 products, and you can see why...***only $399.***

- ► 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

Introducing...

Tomorrow's Disk Controller

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 $399.***

- ► 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.

expected to dominate the PC graphics user-interface market. Further, IBM is well along with TV version II, expected to make the PC look just like an Apple Macintosh, with the icons, mouse and graphics. TV-II should also have enhancements for networking.

Computer Memories Inc. has disclosed that it received an IBM order IBM for 240,000 hard disk drives for the PC/AT to be deliver this year. IBM will no doubt place similar orders with other hard disk drive makers and will also sell many AT's without hard disk drives. Therefore it appears that IBM expects to ship at least a half million AT's this year.

IBM has filed lawsuits agains 11 Taiwan companies charging violation of copyrights on their IBM-PC. There are reports that there are several dozen system copiers operating in Taiwan, most mon-and-pop operations turning out 10 to 20 machines a month. Virtually all of these machines are sold in the far east. Few come to the U.S. as PC clone prices here are already at a point where it is not profitable to sneak these machines into this country. For more comments on this subject see Hank Kee's column in this issue.

Z800 where are you?

About four years ago Zilog disclosed that they were developing the Z800, a souped up version of the Z80. Last year they released preliminary spec sheets for the device. Now they are promising to actually start sampling the device. There is no doubt that 3 or even 2 years ago it would have been a terrific device. Now I wonder.

Not only has Zilog been slow to release the Z800 but they are also way behind on their promised delivery of their Z80,000 32-bit micro. It should be noted that 10-year old Zilog, who is owned by Exxon Corp., has reported 9 un-profitable years of operation. 1983 was their one profitable year. Reportedly, 400 (out of 2100) employees were let go.

Hitachi announces Super Z80

Hitachi has announced a new CMOS 8-bit microprocessor (HD64180) that they claim is compatible with the Z80. They claim the following enhancements: higher speed, 7 new instructions, memory management unit (up to 512K of memory), 2-channel DMA controller, wait-state generator, serial interface, 2-channel 16-bit timer, 12-source interrupt controller, and dual-bus interface. For information: Hitachi Semiconductor, IC Salves and Service Division, 2210 O'Toole Ave, San Jose CA 95131; (408)942-1500.

Phoenix announces XT BIOS ROM

Phoenix Sofware Associates Ltd., 1420 Providence Highway, Suite 101, Norwood MA 02062, (617)769-7020) has announced the availability of a ROM-based BIOS that is totally compatible with the IBM/XT BIOS ROM. Phoenix's IBM/PC compatible BIOS ROM is already being used in such companies as the Tandy Corp., AT&T and KayPro. Phoenix says they will soon announce an IBM/AT compatible ROM BIOS. OEMs can buy an unlimited-use license for $290,000.

Supermicro Newsletter

If you are into the world of multiprocessor, multiuser and networking use of microcomputer systems then you might be interested in knowing about the "Supermicro Newsletter" published by ITOM Interntion Co. Box 1415, Los Altos CA 94022, (415)948-4516. A sample issue is free but a 12 issue subscription is $350 (ouch!). My sample copy was 20 pages long and contained news and articles comparing multi-User SuperMicros to LAN-PCs and a discussion of the leading LAN suppliers.

Thin is in

Planar Systems Inc., Beaverton OR, has announced an electroluminescent flat-panel display with a 4" x 8" display area, displaying 512 x 256 pixels or 80 cols x 25 rows of characters. Moreover, the panel measures 5.7" x 10.3" x .75"... that is right... it is only ¾ of an inch thick. Further, they claim it operates like a conventional CRT utilizing the same sync and video signals, works under virtually all lighting conditions and has a 120 degree viewing angle.

Now for the bad news... current quantity one thousand price is $775. They expect the price to drop to around $250 by 1987. At these prices the old CRT has a lot more life left in it.

Optical disk drives being shipped

Sony and Hitachi are the first companies to ship optical-disk drives for personal computers. The units store 550-Mbytes on a read-only disk the size of a 45-rpm record. Units are expected to start appearing in stores in the last quarter of this year. Panasonic, Matsushita, Phillips, IBM and Digital Equipment Corp. are also expected to announce units this year.

Random news

Lifeboat Associates, 1651 Third Ave, NY NY 10128; (212)860-0300 has released SB-86, a version of MS-DOS 2.11 for use on CompuPro systems using the 8086/8087 or 8085/8088 CPU and Support-I cards. Price is $275... IQ Software, Fort Worth TX has released CP/M-68K for the Apple Macintosh... Statewide Microelectronics Inc., 10 East 22nd St, Lombard IL 60148; (800)882-8311 or 553-7800 has broken the $600 price barrier for a PC 10M hard disk drive system with a $599 package that includes a Cogito 10M drive, controller, cables boot software, mounting hardware, manual and 1 year warranty.

Hitachi and Fuji unveiled, in Japan, cameras that use floppy disks to store the pictures rather than conventional film. The pictures can then be viewed on a TV screen and printed on a thermal color printer in a 3.2 x 5.3 inch format. A 2" disk is used with the drive built into the camera. A workstation is also available which allows editing the pictures, adding titles and random access and auto-search of the disk and interfacing to TV, VCR and a personal computer... AT&T is claiming they will be the first company to manufacture and ship 1-Mbit memory chips. They are promising delivery early next year and are considering selling them to outside customers.

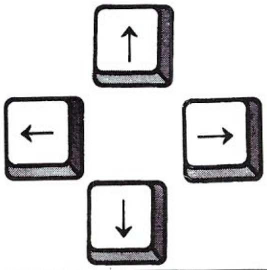
Several companies are introducing boards to allow IBM-AT users to have up to 16 users running under XENIX. One such company we recently learned of is Computone Systems Inc., Atlanta GA shipping a plug-in card called "ATvantge-X". IBM only supports up to three XENIX users on the AT.

Quotation of the month

"IBM's general waffling, wandering, meandering and indecisive smoke generating have slowed down the UNIX market."

Jean Yates
The Yates Perspective, Dec 84
3350 W Bayshort Rd Suite 201
Palo Alto CA 94303

WHEN CONTACTING VENDORS MENTIONED HERE PLEASE TELL THEM YOU READ ABOUT IT IN MICRO/SYSTEMS JOURNAL.



PC/Blue Report

by Hank Kee

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.

The PC/Blue User Group Library is devoted to the collection of public domain software operational on the IBM PC and equivalent clones. This effort is under the sponsorship of the New York Amateur Computer Club and the Amateur Computer Group of New Jersey. Although the bulk of the software is PC-DOS dependent, some of the programs are MS-DOS based and can run with personal computers supporting MS-DOS 2.0 or higher.

The "public domain" software for the IBM PC differs from the concept of public domain programs in the Sig/M (CP/M) User Group Library. Most of the programs in the PC/Blue library are now under the concept of user supported software. Submitted programs are encouraged to be freely copied and distributed. But if you find the programs to be useful, you are requested to send a "donation" to the author. For the most part, these programs are not accompanied by source code.

The large majority of programs in public domain for the IBM PC are well written and very well documented. The topics are wide ranging. This column will highlight recent releases in the PC/Blue library. The following is the contents of the most recently released volumes:

100-101 U.S. Census Utility-County and City Databook

102 Games for IBM-PC
3D Demon Chase
Catch The Bouncing Baby
Castle Adventure
Jumping Janitor Joe

103 Pascal Games
The Third Dimension
3D Graphics Generator

104 Miscellaneous Utilities
Incremental Backup Utility
Library Update written in "C"
Window Pop-up Utility
RAM Disk Utility
Disk Patching Utility
Wordstar/ASCII Reformatter
Library Update written in BASIC
Wordstar 3.0 Notes
Menu-Driven Disk Utility
Hex Converter in MS-DOS
Page Mode List Program
Changing PC-DOS File Attributes
Redirection of Printer Output to Disk

105 Miscellaneous Utilities
"Ultimate" Disk Utility
UNIX Terminal Emulator
Compact Library Update Utilities
dBASE II/III Function Menu
Selection of Screen Attributes
Reminder Signal
Lease/Purchase Option
Incremental Backup BATCH Command
Protect/Unprotect File Utility

106 Modem Programs
PC-Talk III for PCjr
PC-Talk III version 5
IBMODEM modified for PCjr

107 Modem Programs
Kermit version 2.26
QModem version 2.87E

108 Miscellaneous Utilities
Extended Disk Directory v5.4
dBASE Phone Utility
Time Keeper v1.4
Extended Batch Language v2.01b

109-110 Capital PC Remote Bulletin Board System RBBS-PC version 12.3b

A major contribution in the CP/M public domain world were the MODEM and XMODEM programs, written by Ward Christensen way back in 1978. They established a file transfer communications protocol that has become a de facto industry standard. There are now many variants of that program. The fact that Ward made the source code available enable many others to build incremental functionality on to it. There are now many equivalent communication modem programs written for the IBM PC using XModem as the protocol. One of the most popular one is PC-Talk III, written by Andrew Fluegelman. PC-Talk III has evolved very quickly into the equivalent standard for PC-DOS that MODEM is for the CP/M world.

The primary advantage of any program written for the IBM PC is the standardization of the hardware configuration. PC-Talk III needs not to be custom fitted like MODEM and XMODEM since the various hardware assignments have been predetermined by IBM. Add to this the use of a very good screen presentation, PC-Talk III has become a very sophisticated communication program. There are also now versions of it that will run on the PCjr.

A major attraction of PC-Talk III is the availability of the source code. The program was written in IBM BASIC. There are many more people who can understand BASIC than there are who can write programs in assembler format. A compiled version of PC-Talk III can run up to 9600 Baud in addtion to the standard modem speeds of 300 and 1200. Those who have the D. C. Hayes 300 Smartmodem, the user can alternatively run at 450 Baud.

A user's first impression of PC-Talk III is the seeming confusing variety of available commands. The HOME key command however lists all the various options without requiring the user to remember the combination of ALT keys. The documentation is very complete. The only thing lacking as compared with commercial offerings are terminal emulations (e.g. DEC VT-100) and a comprehensive HELP directory.

Like PC-Talk III, RBBS-PC was developed using the IBM BASIC interpreter and compiler. The equipment requirements are minimal. Some of the functions of this system include bulletins, messages, and file utility upload/download.

Another major contribution to the IBM PC community of public domain software is RBBS-PC. The Capital PC User Group, in the Washington D.C. area, has been developing and distributing RBBS-PC. This is a high function remote bulletin board system for the IBM PC. The latest version (12.3b) includes multiuser capability.

RBBS-PC is very responsive as a system. Since it was written in BASIC, it does not, in the true sense have the ability to access files through multiple subdirectories. Messages are all grouped together thereby making access to a "topic" cumbersome. However the source code is there for the more adventuresome.

For more information on contributing or obtaining any of the PC/Blue public domain software write to:

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

EDITOR'S PAGE

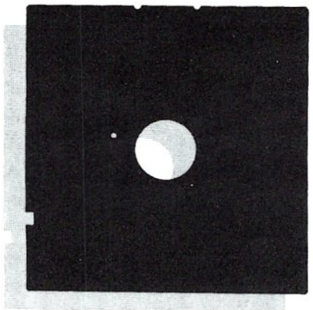
(continued from page 4)

commercial magazine publishing company. But we feel that this is an important magazine and that it needs to be published.

In going back into the magazine publishing business my family and I feel we are better prepared for the demands on our time then we were the first time around. We also realize that we will have to underwrite this endeavor and thus this will really be a labor of love. Hopefully, at some point (in the not too far distant future) we may break even and eventually be compensated for all our work. For this to happen we are counting on a great deal of help from our friends.

If you feel as we do, that there is a real need for this magazine, then please subscribe and tell all your friends to also do so. Also, recommend that companies advertise in MICRO/SYSTEMS JOURNAL and please tell advertisers that you saw their ads in MICRO/SYSTEMS JOURNAL.

I would like to hear your reaction to my comments and to this first issue. Please let me know where you think MICRO/SYSTEMS JOURNAL ought to be going in serving sophisticated micro systems users.



In the SIG/M Public Domain

by Stephen M. Leon

The club meeting is over and the gang has adjourned to the diner for a bite and some bull. One of the old timers laments that everything being written is spoon fed IBM. CP/M is dead and nothing new is or will happen in the Digital Research environment — at least so says our "expert."

It may well be that the eight bit world will never see windows or integrated spreadsheet-databases that draw pictures of fish while communicating with the 82 other machines in their network. If CP/M's fate is an early death much of the fault will lie with Digital Research Inc. While DRI touted the coming of CP/M Plus, SIG/M brought out Rich Conn's ZCPR and ZCPR2 — and they did just about everything Plus did using CP/M 2.2. Had DRI come out with the product on schedule and then supported it, CP/M Plus and not ZCPR would be the 8 bit system of choice.

DRI failed us in eight bit, and what it did in 16 bit borders on a "death wish." In CP/M-86 Plus we saw one of the finest operating systems around. Yet DRI chose not to release it. They have touted Concurrent CP/M under a variety of names, and then failed to promote the product after it was finally released. Watching Concurrent run on a PC and on a CompuPro shows little advantage to using it on the PC, but tremendous potential for the S-100. Yet, while our local DRI office bends over backwards to be helpful, the Pacific Grove group seems to do everything they can to make you go someplace else. It may be easier to purchase top secret CIA reports than it is to buy generic Concurrent for an S-100 — and perhaps even cheaper!

While we can understand some of the problems of DRI and are pleased to see the Japanese companies make their move to CP/M, all is not lost. SIG/M continues to support CP/M, CP/M Plus, CP/M-86 and Concurrent CP/M (or MP/M). Our most recent releases continue this tradition with a volume of CP/M Plus utilities (SIG/M Volume 212) and a CP/M-86 version of Kermit (SIG/M Volume 211). For those of you not familiar with Kermit, it is a modem program developed by Columbia University equally at home on micro and on mainframe. The full SIG/M library will shortly be available for downloading with Kermit from a mainframe in Somerset, New Jersey.

On the 16 bit side, we are trying to release software that will operate under plain CP/M-86 as well as Concurrent. Volumes 216 and 217 released in January 1985 contain the latest version of Modem 7 for CP/M-86. Earlier releases had contained translations of much simpler Modem 7 versions. This new release is a full feature modem with auto dial and all of the other extras found in the latest versions. Further, it supports both the CompuPro and the Gifford versions of MP/M. We have often heard users of the 8-16 multi-user system complain that they could not get a modem program to work on their system. That software is now available free.

Also on Volume 217 is a Z80 emulator for CP/M-86. With this program you can simulate a Z-80 environment on an 8086 machine. The same type of cross-fertilization is attempted on Volume 212 with a program to read MSDOS disks under CP/M 80. Often we have bought commercial software only to find it did not perform as anticipated. The same holds true with public domain software, except that in most cases SIG/M also distributes the source code. As improvements and fixes are made, new versions are issued.

We were unable to get the MSDOS disk reading program to work but released it anyway with a request that somebody fix it and send it to us. (Same thing happened with the Z-80 emulator on Volume 217. We released it last year with a request that someone get it to work, and Bill Earnest did just that.)

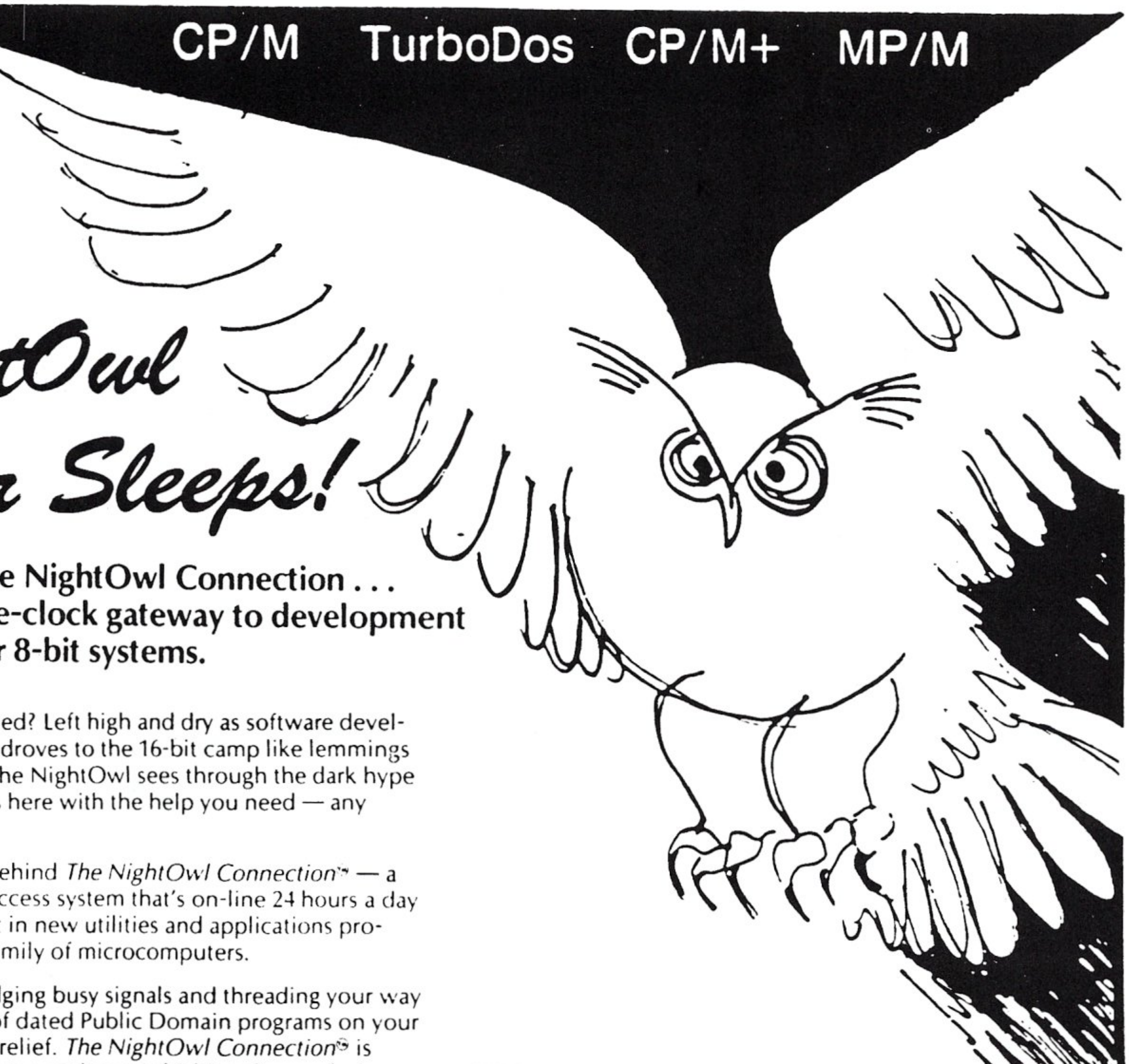
In 1984 SIG/M received its first contribution from Professor Harold McIntosh from the Universidad Autonoma de Puebla, Mexico. It was called REC, or Regular Expression Compiler, and was a "compiler compiler." Using braces and plus and minus symbols, REC wrote beautiful 8080 and 8086 programs. We asked Dr. Andy Bender, SIG/M's resident compiler expert, to try REC and he just went wild. REC is not for the novice! It is for those with a craving to learn who already have basic skills. It can open up a new world of programming. An updated REC in 8080 and 8086 with and without floating point math is available on Volumes 213 and 214. On volume 215 we have new REC documentation and a new version of CNVRT, a REC program that simplifies REC programming.

If the concept of REC is obscure to you, take a look at the product it produces. Using REC, Professor McIntosh has written RUN (SIG/M Volume 215) which allows you to run a CP/M 80 or CP/M 86 file (squeezed or unsqueezed) located in a library. For more information about the many REC volume and programs written using REC, check the new January 1985 SIG/M catalog (covering Volumes 1 through 217).

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 distributor list is included with the printed catalog. A disk version of the catalog (Volume 00) 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.

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.



The NightOwl Never Sleeps!

Introducing The NightOwl Connection . . . your 'round-the-clock gateway to development and support for 8-bit systems.

Feeling abandoned? Left high and dry as software developers defect in droves to the 16-bit camp like lemmings rushing to the sea? The NightOwl sees through the dark hype and murky fads. He's here with the help you need — any time you need it.

That's the concept behind *The NightOwl Connection*™ — a multi-user, remote access system that's on-line 24 hours a day offering you the best in new utilities and applications programs for the 8080 family of microcomputers.

If you're tired of dodging busy signals and threading your way through a haystack of dated Public Domain programs on your local RCP/M, here's relief. *The NightOwl Connection*™ is your complete, on-line warehouse of 8-bit software, featuring:

- The latest in public domain programs: ZCPR (1, 2 & 3) plus enhancements; Languages, including a complete Falconer PascalP library; Utilities; Tool boxes; Applications; even an 8080-to-8086 translator.
- Support for C, Pascal, WordStar, dBASE II and BASIC.
- A total of 36 separate user directories, two of them dedicated to Kaypro and Osborne software.
- A message and conference system that allows you to communicate with other subscribers and seek direct help from NightOwl's programmers, system developers and customer service department.

Best of all, the *Connection* gives you something no other remote access system can ever offer: a treasury of programs for CP/M, CP/M 3 and TurboDos operating systems developed exclusively for our subscribers *plus* a 10 percent discount on all purchases of proprietary software released by NightOwl Software, Inc.

We're the developers of MEX 1, the communications package that set modems dancing on their serial cables last year. You'll find more MEX 1 support — from latest overlays to the most sophisticated command files — on the *Connection* than anywhere else. But MEX 1 is only the beginning.

This year, we're offering a host of new programs, including two hot utilities:

CAPTURE, a CP/M traffic controller, can instantly redirect the output of most programs from screen to printer, from printer to disk, from disk to screen — you name it!

GENPATCH, a debugging aid that modifies the TurboDos 1.3 GEN program to produce MAC and SID compatible symbol files.

. . . and then there's *MEX II*™, the program that redefines telecomputing. If you think MEX 1 is powerful, wait til you experience the thunder of *MEX II* with its conditional command executions, compacted subroutine design — and unlimited expandability! A strictly proprietary communications system, *MEX II* is currently undergoing beta testing, and is due to be released early this spring at a price of $99.95.

Normally, subscribers to *The NightOwl Connection* would be entitled to a 10 percent discount on *MEX II*. But for a limited time, we're offering a special introductory package: For the price of the *Connection*'s standard sign-up fee of $100, new subscribers will receive a one-year subscription to the system *plus* a fully documented copy of *MEX II* at no extra cost. Subscriptions can be renewed for $50 a year.

NightOwl Software, Inc.

To order outside Wisconsin, call 1-800-NITEOWL. In Wisconsin, call 414-563-4013.
Or write: NightOwl Software, Inc. Route 1, Box 7, Fort Atkinson, WI 53538

master charge VISA

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

Volume 210 CP-M 86 Utilities
released January 18, 1985

BLIST	.LBR	multiple file printer
CLCIB86	.LBR	CB86 command line interpreter
CRC	.CMD	CRC checker in CP-M 86
CRC-64	.AQ6	/
CRC68K	.LBR	CRC checker in CP-M 68K
CRCBUILD	.LBR	builds catalog files — CP/M 80 & 86
PUT	.AQ6	copies files between user areas
PUT	.CMD	/
SQ183	.CMD	CP-M 86 file squeezer
TAB86	.LBR	creates tabs in code
UNTAB86	.LBR	removes tabs from code
USQ	.CMD	CP-M 86 file usqueezer

Volume 211 Kermit MODEM for CP-M 86
released January 18, 1985

86KERMIT	.LBR	194K Kermit modem for CP-M 86

Volume 212 CP-M Plus Utilities & Misc. Routines
released January 18, 1985

CHIS16	.CQ	Chi-square calculation in C
CPM2-+	.LBR	allows 2.2 routines to run under CP-M Plus
CPM3-CAT	.LBR	CP-M Plus catalog program
CPM3LIB	.LBR	CP-M Plus subroutine library
CURLY	.LBR	checks braces in C programs
DAYS	.CQ	changes 01/18/85 to 1.18.1985
FIND+2	.LBR	updates FIND
KPROFMT	.LBR	multidisk formats for Kaypro 2, 4 & 10
QS-CPM3	.LBR	resets disk attributes in CP-M Plus
RDMSDOS	.LBR	reads MSDOS disks in CP-M — maybe ?
SUPERZAP	.LBR	Z80 disk utility similar to DU
UNERA+	.LBR	UNERASE in CP-M Plus
YAM	.H	file missing from SIG-M 183

Volume 213 Regular Expression Compiler (REC) (Volume 1 of 2)
8080 Floating Point, 8086 w/o Floating Point Math
Universidad Autonoma de Puebla, Mexico
released January 18, 1985

REC80F	.LBR	8080 REC with floating point math
REC86	.LBR	8086 REC w/o floating point math

Volume 214 Regular Expression Compiler (REC) (Vol 2 of 2)
8086 Floating Point, 8080 w/o Floating Point Math
Universidad Autonoma de Puebla, Mexico
released January 18, 1985

REC80	.LBR	8080 REC w/o floating point
REC86F	.LBR	8086 REC with floating point

Volume 215 REC Documentation, Updated CNVRT, RUN —
executes files from SUBMIT files or libraries
Universidad Autonoma de Puebla, Mexico
released January 18, 1985

CNVRT	.LBR	latest version of CNVRT
HJELP	.COM	examine libraries as HELP files
RECDOC	.LBR	manuscript describing REC
RUN	.LBR	runs 80 & 86 programs from SUB or LBR files

Volume 216 MODEM (MDM840) for CP-M 86 and MP-M
(Volume 1 of 2)
released January 18, 1985

MDM840	.AQ6	MDM Modem series translated for CP-M 86
MDM840	.DQC	documentation file
MDM840	.HQ6	hex file
MDM840	.NQT	translator's note
M8LIB	.AQ6	telephone auto dialer library for MDM840
M8LIB	.CMD	/
M8LIB	.DOC	/
M8NM-40	.AQ6	/

Volume 217 MODEM (MDM840) for CP-M 86 and MP-M (Vol 2 of 2)
CP/M 80 Emulator; dBASEII Toolkit
released January 18, 1985

M8GP-1	.AQ6	MDM840 general purpose overlay
M8IN-1	.AQ6	CompuPro Interfacer ¾ overlay
M8MCDOS	.AQ6	CompuPro Interfacer ¾ overlay
QUEIN	.AQ6	with Gifford MC-DOS
M8MPMIN	.AQ6	CompuPro MP-M 86 with MSUP.RSP
MSUP	.AQ6	/
MSUP	.DQC	/
M8RB-1	.AQ6	DEC Rainbow overlay
Z80	.A86	working CP-M 80 emulator for CP-M 86
Z80	.CMD	/
CBADDR	.Z80	/
CODE8086	.Z80	/
CODECB	.Z80	/
CODEED	.Z80	/
EDADDR	.Z80	/
JUMPADDR	.Z80	/
FUNCLIB	.CMD	dBASEII toolkit library
DATELIB	.CMD	/
MATHLIB	.CMD	/
TOOLSLIB	.DOC	/
LOGOFF	.ASM	bulletin board security system
LOGOFF	.COM	/
LOGON	.ASM	/
LOGON	.COM	/
LOGON	.DOC	/
LUU	.CQ	Australian version of CP-M 86 LU
LUU	.CQD	/

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™	$695	Lomas Lightning 286™	$821	MACROTECH 286/Z80H	$995	TELETEK SYSMASTER II	$899
CompuPro MDRIVE-H®	$495	Lomas THUNDER 186™	$1195	EARTH TURBO SLAVE I	$399	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	$459	CompuPro 286™	$695	CompuPro RAM 22™ 120ns	$ 995	CompuPro System Support 1™	$297
CompuPro Disk 3™ A&T	$525	CompuPro SPU-Z™	$261	CompuPro Ram 23™ 64K 120ns	$ 308	CompuPro Interfacer 4™	$297
LOMAS LDP 72 Floppy Contr	$220	CompuPro 8085/88™	$327	CompuPro RAM 23 128K 120	$ 571	CompuPro Interfacer 3™	$459
Lomas WD1002 HD Contr	$325	CompuPro CPU Z™	$215	Lomas RAM 67 128K 100ns	$ 725	Lomas 2 ser, 2 par, clock	$275
Teletek FDC II Flpy cont	$245	Lomas Lightning 286	$821	Lomas Megaram 256K 150 ns	$ 476	Lomas Octaport 8 serial	$316
Teletek HD/CTC HD/Tape	$525	Lomas 8086 8MHz	$420	Lomas Megaram 512K 150ns	$ 876	Lomas Octaport 4 serial	$200
80287 FOR LOMAS 286	$395	Lomas 8086 10MHz	$520	Lomas Megaram 1024K 150ns	$1596	Teletek PS10 2 par, 4 ser	$215
80287 FOR COMPUPRO 286	$325	Macrotech 286/Z80	$995	Lomas Megaram 2048K 150ns	$2995	Teletek I square 1 port	$195

SINGLE BOARD COMPUTERS		HARD DISKS		FLOPPY DRIVES		S-100 COLOR GRAPHICS	
THUNDER 186 256K QTY 1	$1195	Rodime 13/10 90 ms	$ 695	8" Mitsubishi 2894-63	$389	Lomas Color Magic 16K	$476
THUNDER 186 256K 3 UP	$1095	Rodime 27/21 90 ms	$ 895	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	$ 695	5" Dbl. Sided Color Diskettes 10 Pk	$ 25	runs Auto-CAD-Z & other on Lomas	
EARTH TURBO SLAVE I 8MHZ	$ 399			5" Dbl. Sided Color Diskettes 100 Pk	$240		

LOMAS MARCH ONLY BOARD SETS 8086, 512K Megaram, LDP72, Hazitall & CCP/M $1748. Same but 286 instead of 8086 $2135.

SYSTEMS READY TO RUN

VALUE ADDED RESELLERS SPECIAL

Lomas 286, 1024K, 6 serial, 2 Centronics, 20 MB Hard Disk, 1 - 5" Floppy Multi Concurrent Dos	
in 15 Slot Desk Mount Cabinet	$4999
or in 20 Slot Pronto Roll A Round All Constant Voltage Cabinet only	$5695
Options 40 MB HD, 80287, 10 Serial instead of 6, 24 MB Tape, 2nd Floppy	
Up to 2-8" and 2-5" Floppys. The 15 Slot cabinet 20 MB at $4999 is best buy.	
THUNDER 186 256K, 2-5" Floppys, 4 Slot cabinet 110/220 volt, Concurrent DOS®	$1795
THUNDER 186 512K, 1-5" Floppy, 4 Slot cabinet 4 Serial, 3 Centronics, 20 MB Hard Disk Multi-user Concurrent Dos®	$3795
Lomas 286 CPU, 1024K Megaram, 6 serial, 2 centronics, 15 Slot, 40 MB Hard Disk, 1-5" Floppy Multi-Concurrent Dos®	$5799
CompuPro 286, 512K Ram 22's, System Support 1, I/O, 15 Slot, 20 MB Hard Disk, 1-5" Floppy Multi-Concurrent Dos®	$6095
CompuPro 286; 1024K 120ns Dram, System Support 1, I/O 4, 15 Slot, 40 MB Hard Disk, 1-5" Floppy Multi-Concurrent Dos®	$6395
CompuPro 8085/88, Disk 1A, Interfacer 4, Ram 23 64K, 2-8" Double Sided Floppys, CP/M® 2.2, 10 Slot Cabinet	$2895
CompuPro CPU Z, Disk 1A, Interfacer 4, Ram 23 64K, 2-8" Double Sided Floppys, CP/M® 2.2, 10 Slot Cabinet	$2795
CompuPro 8085/88, Disk 1A, System Support 1, Interfacer 4, Ram 23 128K, 2-8" DSDD Enclosure 2 & DR Cab. CP/M 8-16	$3995
Teletek Systemaster II, 8MHz Z80H 128K, Single user Turbodos ™ w/spooler, 2-8" DSDD Floppys 10 Slot canbinet	$2695
Teletek Systemaster II, Multi-user Turbodos™, 1-8", 20 MB hard disk, 4 Hi Speed Slaves 128K, 10 Slot Cabinet	$5695
Teletek Systemaster I, CP/M 2.2, 2-8" DSDD FLOPPIES, 10 Slot Cabinet	$2095

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

Independent, Custom, Scientific Programming Service Available

CABINETS		TERMINALS		PRINTERS	
CompuPro Enclosure 2™ Desk	$611	Qume 102 Amber 14"	$495	DATA PRODUCTS 8010	$495
CompuPro Enclosure 2 Rack	$645	Qume 102 Green 14"	$490	DATA PRODUCTS 8020	$625
Para Dynamics CVT Rack DR Cab 2-8" Desk	$395	AMPEX 230	$549	C. ITOH 8510 PARALLEL	$350
Para Dyn CVT Rack DR Cab 2-8"	$495			C. ITOH 8510 SERIAL	$455
Para Dyn CVT Desk DR Cab 1-8", 5" HD	$425	**SOFTWARE**		C. ITOH 8510 SP	$475
Para Dyn Roll A Round CVT Pronto	$1195	CompuPro Concurrent Dos®	$250	C. ITOH 8510 SR SERIAL	$550
Para Dyn 10 Slot Mini Pronto CVT	$795	CompuPro CP/M® 8-16™	$175	C. ITOH 1550 PARALLEL	$550
Hard Disk 5" Floppy & Hard Disk Cab	$175	Lomas MSDOS™ 2.11	$200	C. ITOH 1550 SR 180 CPS SERIAL	$725
Two 5" Hard Disk Cabinet	$195	Lomas Multi Concurrent Dos™	$360	C. ITOH 1550 SP 180 CPS PARALLEL	$695

ALL PRICES & SPECIFICATIONS SUBJECT TO CHANGE AND STOCK ON HAND

The C Forum

by Don Libes

What a relief that C is such a "small" language. Few keywords or restrictions to worry about. No complex operators or datatypes to mishandle. The language is surprisingly svelte for all its power. But while you were able to pick up a book on C and breeze through the first half, you may never have gotten very comfortable with it, even if you're fluent in another programming language.

Why? C abounds with subtleties that are written between the lines in the UNIX manuals and most of the C language books that I've seen. One of the best places to really see these brought out are the experienced C programmer's code. They know all the tricks. If they were nice enough to leave comments, we might learn them also!

As UNIX is being distributed more and more in binaries form, it is difficult to learn from UNIX source code samples. Fortunately, many books have come out to fill the void and since they were written for the purpose of education, you would think they could do the job a lot better. But most of them slanted towards an introduction, while other are references. Very few discuss intermediate-level topics in detail. That is what this column will cover.

You are encouraged to write to me about topics or problems that you want to know about. I want this column to be reader driven. Until it is, I will write about topics that I'm sure are of general interest. I just happen to have one here!

Variably-Sized Arrays

If you've ever written generic subroutines that operate on arrays, you've probably run across the problem of trying to pass arrays of arbitrary sizes. For example, suppose we would like to have a routine that prints out matrices of arbitrary sizes. We start out:

```
print_array(array)
int array[][];
```

The C compiler doesn't accept this. (Mine prints out "null dimension".) This is because arrays are stored without information such as the number of rows and columns. Well, then lets try adding the size of the array to the parameters.

```
print_array(array,r,c)
int array[r][c];
```

The C compiler rejects this also, saying "constant expected". Variable sizes must be constant in C. What a pain!

There are several ways of getting arbitrarily sized arrays, however.

One way is to do the addressing yourself. With the help of a macro, this solution is readable.

```
#define MAT(x,y)        mat[x*c + y]

print_matrix(mat,r,c)
int *mat;
int r, c;
{
        int i, j;

        for (i = 0; i < r; i++) {
                for (j = 0; j < c; j++) {
                        printf("%d ",MAT(i,j));
                }
        putchar('\n');
        }
}
```

Now, we can declare the matrix and call our routine as follows:

```
int matrix[ROWS][COLUMNS];

print_matrix(matrix,ROWS,COLUMNS);
```

The main drawback to this solution is that it's time-expensive. Each time you reference the array, you perform a multiplication and addition. Thus, to access every member in the array requires ROWSxCOLUMNS multiplications. The other drawback is that the macro MAT requires the number of columns being available. We can do better.

You may never have realized that it isn't necessary to perform those multiplications, simply because it seems inherent in figuring out matrix element addresses. However, if we are willing to sacrifice some storage we can avoid the multiplication.

What we do is calculate the addresses for the base of each row once and store them in a separate (one-dimensional) array. Then we can get to any element simply by adding the column offset to the base address of the appropriate row.

For example, a 5x3 array would require a 5 element "dope vector". Each element of the dope vector is an address of 3 elements of the array.

Since we can get to every element of the matrix through the dope vector, there is no need to pass the array itself. So the first argument becomes the dope vector. (Its still called "mat" though.)

Now print_matrix() looks like this.

```
print_matrix(mat,r,c)
int *mat[];     /* dope vector: array of pointers to ints
                */
int r,c;        /* rows and columns */
{
        int i, j;

        for (i=0;i<r;i++) {
                for (j=0;j<c;j++) {
                        printf("%d ",mat[i][j]);
                }
                putchar('\n');
        }
}
```

This type of array takes a little more work to set up:

```
int *matrix[ROWS];          /* this is the dope
                               vector */
int i;

/* now initialize each pointer in the dope
vector */
for (i = 0 ; i < ROWS ; i++ ) {
        /* allocate space for the columns */
        matrix[i] = (int *) malloc(COLUMNS *
                                  sizeof(int));
}

print_matrix(matrix,ROWS,COLUMNS);
```

This technique has the disadvantages that it takes up a little more space

(continued on page 19)

C FORUM

(continued from page 18)

than a true array and it requires initialization (though you can create a subroutine to do this for you).

But the advantages are many. It's faster than real arrays because no multiplication is performed to do addressing. Each row can have a different number of elements. Applications of this are to store different length strings in an array or keeping an open hash table.

This technique also extends to higher dimensioned arrays (where time savings become even better). Also, these arrays can be created dynamically. A final goodie that I'll mention is that by adjusting the dope vector (or the pointer to it), it's possible to get 1-based (or any number) indices rather than 0.

Don Libes
seismo!nbs-amrf!libes

Don Libes is a computer scientist working in the Washington DC area. He works on artificial intelligence in robot control systems. He is also the son of Lennie and Sol Libes.

Bringing up CP/M Plus on your System

by Sheldon Kolansky

Many of us have been watching Digital Research tout the new version of CP/M-80. The features look so nice, and they are. The buffering should make it run much faster, etc. Well, I went out and bought the new version and worked for several weeks getting it going. I would like to pass on my trials and tribulations, maybe I can make it easier for you (if you should decide to do it).

The Digital Research manuals are getting better, but some areas leave a lot to be desired. The recommended first and most important step is to bring up the simplest system possible. My first system started with just a single density disk and only console I/O.

The BIOS provided is broken up into several modules which can be assembled separately and then linked together. I modified the appropriate modules and tested the linked BIOS to my satisfaction. CP/M 3 BDOS is provided in relocatable format, and your BIOS must be linked to it.

After the entire system is linked together, a program called CPMGEN must be executed. CPMGEN asks you for pertinent information via a question and answer session at the terminal. This information includes how much memory you have, terminal parameters, etc. The output of CPMGEN is CPM3.SYS, the actual operating system image.

Here are two tidbits of important information: First, make your CP/M 3 system reside under SID loaded via your current system. This will allow you to debug the system using SID. Secondly, the sample BIOS provided assigns space for the allocation vectores of the DPH. GENCPM asks if double allocation vectors are to be used. If your answer to GENCPMs question is Yes, then be doubly sure you have allotted sufficient space in the allocation vector.

Then I tried to figure out how to boot it. The DRI manuals talk about CPMLDR, which reads into memory by a hardware boot, and is supposed to read the system image off the disk and put it in memory and execute it. If you read closely you will find out that you must write a LDRBIOS, but who knows what a LDRBIOS is? the LDRBIOS is a basic BIOS that performs limited functions of your regular BIOS for the sole purpose of reading a file (CPM3.SYS) into memory and executing it. The LDRBIOS is linked to CPMLDR.REL to form CPMLDR.COM. CPMLDR.COM is read into memory by whatever boot process you intend to use. CPMLDR can also be executed as a file under your current operating system; this makes it much easier to debug.

One area of the manuals that leaves a lot to be desired is the description of the LDRBIOS. DRI manuals present about a page of "information" on the LDRBIOS. Well, after many hours of playing, I determined that the only calls used by CPMLDR are: BOOT, CONOUT, SELDSK, HOME, SETTRK, SETSEC, SETDMA, SECTRN, and READ. I formed my version of the LDBRIOS by combining the BIOSKRNL, CHARIO, and my disk module together in one file and deleted all the extraneous "nice" features, and RETurned calls that wouldn't be used. I did not include the SCB as it isn't necessary. The DPH macro that the sample BIOS calls cannot be used because it relies upon GENCPM to allocate buffers, and GENCPM is used on generating the operating system only, not the LDRBIOS. You have to allocate buffers for the directory, data, check vector and allocation vector (I guessed on the last two). This reduced the size of the code immensely. The basic single density system worked after a couple of minor bugs were squashed.

The single density system worked well. Be sure to test it completely. In the later steps I wished I had not blown away this version of the system, which would have have let me go back and test certain features with which I was having trouble.

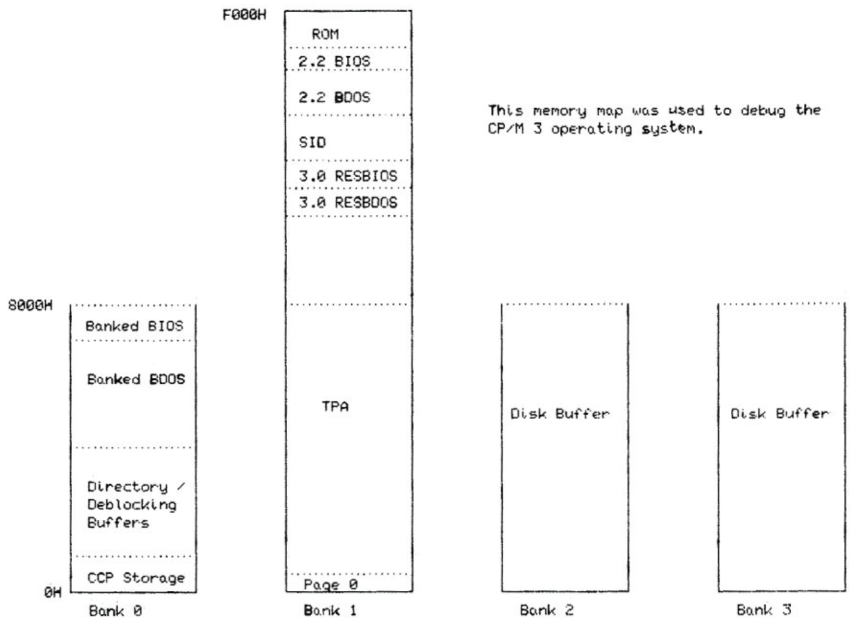
I got brave and decided to get double density working with automatic density selection, and that's when all my problems started. The following notes should outline some of my solutions/problems with the LDRBIOS.

MOVE: The call to MOVE is required if the CPMLDR is to do the built in blocking/deblocking.

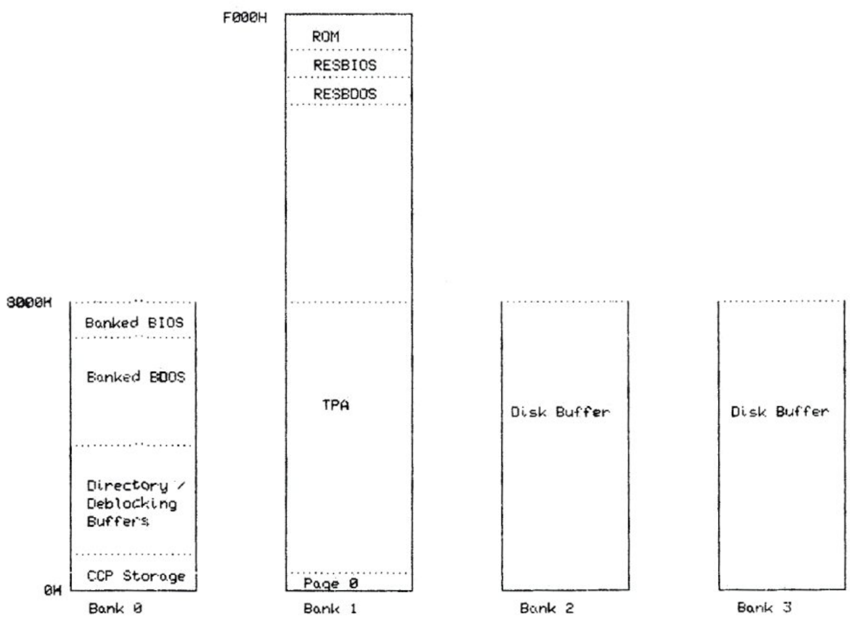
SECTRN: must be modified if no translation is required. CPMLDR or BDOS reads sectors starting at 0 and relies upon SECTRN to provide a legal sector number for the disk. A no-translation-required routine, as usually provided with the sample BIOS, will pass

Table 1. Benchmark Tests

Command Line > System	MAC FMT	PMATE FMT.ASM	TYPE FMT.ASM [NOPAGE]
V2.2	1:47	9.8	44.5
V3.0			
NB	1:46	9.8	45.0
S1	1:46	12.9	49.9
S3	1:38	12.8	50.3
RD-A	1:49	13.0	49.8
RD-C	1:10	7.7	46.0



Memory Map



Memory Map

```
;LAST EDITED 07/06/83 12:20:40
;ADDED DOUBLE DENSITY SUPPORT
;LAST EDITED 07/02/83 16:09:42
;LDRBIOS   BIOS FOR CPM3 CPMLDR
;
MACLIB   CPM3
;
;  DISK READ AND WRITE ENTRY POINTS.
;
;JADE DOUBLE DENSITY DISK CONTROLLER
;
D$PORT   EQU      043H     ;DOUBLE D PORT ADDRESS
D$ADDR:  EQU      0FC00H   ;ADDRESS OF DD BUFFER

;  DOUBLE D HARDWARE COMMANDS

DC$MB0   EQU      01H      ;SELECT DOUBLE D BANK 0
DC$MB1   EQU      03H      ;SELECT DOUBLE D BANK 1
DC$SOT   EQU      0        ;SWITCH DD MEM OUT OF SYSTEM
DC$INT   EQU      02H      ;ISSUE DD Z80A INTERRUPT
```

sector 0 to the floppy, and since most floppies start with sector 1, this causes a problem. This same "problem" existed with 2.2 if you did the deblocking. You can get around it by incrementing the sector number by one if no translation is required.

DIRBCB: I didn't know what the values for sector, etc. would be, so I arbitrarily assigned 0 to all except the buffer address. Boy, was that a mistake! The blocking/deblocking algorithm of the CPMLDR went to read the first sector of the directory, checked the DIRBCB and decided that the sector had already been read since the bytes described block 0, the first sector. CPMLDR then went and searched the directory buffer for CPM3.SYS, didn't find it, and then read the next directory sector. Therefore if CPM3.SYS was in the first directory sector CPMLDR could not find it. The answer to this problem took four calls to DRI (I dialed about 400 times to get through four times). Set the first byte of the DIRBCB to OFFH and this will unallocate the buffer.

SID is provided with CP/M 3, and is a handy debugging tool, but could use some new features to work with relocatable files. It is difficult at best to determine the location of the code segment that you're working on, and when you do find it, SID does not allow you to enter it as an offset for all future address references. SID is described in the manuals two or three times and DRI provides a handy SID command summary card; only one of these descriptions is correct—table 5-18 of the Users Guide. The old SID required:

```
Ifilename.typ
Roffset
```

to read a .HEX file. The new SID uses:

```
Rfilename.typ, offset
```

to read either a .COM or .HEX file. I like the new SID better, but think the manuals should reflect how it works.

LINK is also provided with CP/M 3. It does a fine job providing you write excellent code the first time and don't have to find it for debug. You should become familiar with the HEX math commands in SID.

In version 2.2, if you wished to redirect the console or printer you built the feature into your BIOS and used the IOBYTE to control it. DRI has been nice enough to reserve location 3H for the IOBYTE, but no longer supports the IOBYTE function and has replaced it with DEVICE. DEVICE is much better than IOBYTE, in that it is much more flexible, and allows more choices.

But my boot ROM had the IOBYTE support blown in. I strongly recommend that IOBYTE support in ROM be removed if you would have a banked system. If not, you must limit the buffer area in bank 0 which may be in context whenever character I/O is done, since using the buffer may change the contents of the IOBYTE. Upon initialization you must now init the IOBYTE in two different banks, 0 and 1. Failure to use caution will result in your system trying to talk to alternate devices and mislead you into wondering where your system went.

The version 1 and 2 SYSGEN program has been replaced by COPYSYS. I have found two bugs in COPYSYS. COPYSYS will read a .COM file and place it on the system tracks. One bug related to reading this file under version 3.0. COPYSYS calls BDOS function 50 (direct BIOS) to set the DMA address and should have called BDOS function 26 (set DMA address) while reading the file, and function 50 when reading or writing the system tracks image. The other bug appeared when I wrote the track image under 3.0 to a 1K sector track. CP/M was doing the blocking and deblocking, and COPYSYS never told it to flush the buffer, therefore it never wrote the last sector to the disk. This can be fixed by calling BDOS function 48 (Flush Buffers) at the end of the copy.

In trying to make life simple, I wrote the file shown in figure 2 to test COPYSYS, and also to be sure I was placing the binary code in the proper place on the system tracks. As you can see the first track is single density and the second track is double. I assembled the file, used SID to load it over a field of zeros, and then saved it. After writing it to the system tracks, I used a dump program running under version 2.2 to see that each sector contained data that verified its address. I also read this file into memory in order to easily determine what offsets to use in loading the files that made up the system tracks image.

I built many versions of the system as each new feature was incorporated. All the features that could be incorporated in the non-banked system were implemented before I continued on to banked systems. When I started working with banking, my common area was a 32K segment from 8000H up. I had four 32K banked memory segments initially which increased to 10 by the completion of this article. The 32K common area allowed me to keep the version 2.2 system I was operating under, the version 3.0 system being debugged, and SID in common memory. The progression followed to 3 extra banks, and then

LISTING #1

```
; DISK CONTROLLER MODULE LINKAGE (DCM - VER 2.2)

;( DCM ADDRESSES DEFINED )

DD$CBT   EQU     0370H+D$ADDR        ;COMMAND BLOCK    (BANK 0)
DD$SBUF  EQU     0380H+D$ADDR        ;SD SECTOR BUFFER   (BANK 0)
DD$DBUF  EQU     0+D$ADDR            ;DD SECTOR BUFFER (BANK 1)
DD$DPB   EQU     03A0H+D$ADDR        ;ID SEC DPB        (BANK 0)
DD$DDF   EQU     03B1H+D$ADDR        ;ID SEC FLAGS      (BANK 0)

;( DCM COMMANDS )

DC$LOG   EQU     000H     ;LOG ON DISKETTE
DC$RDS   EQU     001H     ;READ SECTOR
DC$WRS   EQU     002H     ;WRITE SECTOR
;
DPB$SZ   EQU     15       ;BYTES TO DISK PAR HEADER
ROM      EQU     0F000H
;
MONIT    EQU     ROM
BLOCK:   EQU     ROM+36   ;BLOCK MOVE ROUTINE(LDIR)
;
CR       EQU 13
LF       EQU 10
?BOOT:   JMP INIT          ; INITIAL ENTRY ON COLD START
?WBOOT:  JMP STOP          ; REENTRY ON PROGRAM EXIT, WARM START

?CONST:  JMP STOP          ; RETURN CONSOLE INPUT STATUS
?CONIN:  JMP ROM+0CH       ; RETURN CONSOLE INPUT CHARACTER
?CONO:   JMP ROM+0FH       ; SEND CONSOLE OUTPUT CHARACTER
?LIST:   JMP STOP          ; SEND LIST OUTPUT CHARACTER
?AUXO:   JMP STOP          ; SEND AUXILLIARY OUTPUT CHARACTER
?AUXI:   JMP STOP          ; RETURN AUXILLIARY INPUT CHARACTER
?HOME:   JMP HOME          ; SET DISKS TO LOGICAL HOME
?SLDSK:  JMP SELDSK        ; SELECT DISK DRIVE, RETURN DISK INFO
?STTRK:  JMP SETTRK        ; SET DISK TRACK
?STSEC:  JMP SETSEC        ; SET DISK SECTOR
?STDMA:  JMP SETDMA        ; SET DISK I/O MEMORY ADDRESS
?READ:   JMP READ          ; READ PHYSICAL BLOCK(S)
?WRITE:  JMP STOP          ; WRITE PHYSICAL BLOCK(S)

?LISTS:  JMP STOP          ; RETURN LIST DEVICE STATUS
?SCTRN:  JMP SECTRN        ; TRANSLATE LOGICAL TO PHYSICAL SECTOR

?CONOS:  JMP STOP          ; RETURN CONSOLE OUTPUT STATUS
?AUXIS:  JMP STOP          ; RETURN AUX INPUT STATUS
?AUXOS:  JMP STOP          ; RETURN AUX OUTPUT STATUS
?DVTBL:  JMP STOP          ; RETURN ADDRESS OF DEVICE DEF TABLE
?DEVIN:  JMP STOP          ; CHANGE BAUD RATE OF DEVICE

?DRTBL:  JMP STOP          ; RETURN ADDRESS OF DISK DRIVE TABLE
?MLTIO:  JMP STOP          ; SET MULTIPLE RECORD COUNT FOR DISK I/O
?FLUSH:  JMP STOP          ; FLUSH BIOS MAINTAINED DISK CACHING

?MOV:    JMP MOVE          ; BLOCK MOVE MEMORY TO MEMORY
?TIM:    JMP STOP          ; SIGNAL TIME AND DATE OPERATION
?BNKSL:  JMP STOP          ; SELECT BANK
?STBNK:  JMP STOP          ; SELECT BANK FOR DISK I/O DMA OPERATIONS.
?XMOV:   JMP STOP          ; SET SOURCE AND DESTINATION BANKS


STOP:    RST      7        ;NOT IMPLEMENTED
;
PMSG:                      ; PRINT MESSAGE @<HL> UP TO A NULL
                           ; SAVES <BC> & <DE>
         PUSH B
         PUSH D
PMSG$LOOP:
         MOV A,M
         ORA A
         JZ PMSG$EXIT
         MOV C,A
         PUSH H
         CALL ?CONO
         POP H
         INX H
         JMP PMSG$LOOP
PMSG$EXIT:
         POP D
         POP B
         RET
```

```
; SELDSK
SELDSK: XRA     A          ;SET FOR DRIVE ZERO
        STA     @RDRV
LOGIN:  LXI     D,LOG$MSG
        SHLD    OPERATION$NAME
        LXI     H,FDSD0 ;POINT TO DRV 0
LGIN1:  SHLD    DT$PTR  ;SAVE FOR LATER
        CALL    BNK0    ;SWITCH TO BANK 0
        CALL    PMOVE   ;SET UP COMMAND BLOCK
        MVI     A,DC$LOG        ;LOAD DCM LOG-ON CMND
        CALL    DSK$EX  ;PERFORM DISK OP
        JZ      LOG$CK  ;GO TO LOGON DISKETTE
        LXI     H,0     ;ERROR, BAD LOG ON
        JMP     DSK$ER  ;BIOS EXIT

SETDD:  LXI     H,1024  ;MUST BE DOUBLE
        SHLD    SEC$SZ
        LXI     H,DD$DBUF       ;1K SECTOR BUFFER ADDR
        SHLD    DD$BUF
        JMP     BNK1
;
;
;( CHECK FOR JADE ID )*

LOG$CK: LHLD    DD$BUF  ;DD BUFFER
        LXI     D,JADEID        ;DE PNTS BIOS ID
        MVI     B,ID$SZE        ;SET LABEL SIZE
LOG$ID: LDAX    D!      INX D   ;GET LABEL CHARACTER
        CMP     M!      INX H   ;DOES ID SECTOR MATCH
        JNZ     TR3740  ;ASSUME DISKETTE 3740
        DCR     B       ;DECREMENT COUNT
        JNZ     LOG$ID  ;CHECK IF ANOTHER CHR

;( DISKETTE CONTAINS ID )

;( SET NO SECTOR TRANSLATION )

TRNONE: LXI     D,TRANS1
        LHLD    DT$PTR  ;ADDR OF PARA HDER
        MOV     M,E     ;SET LOW ORDER ADDR
        INX     H       ;NEXT BYTE
        MOV     M,D     ;SET HIGH BYTE
;
;GET PARAMETER BLOCK FROM DD BUFFER AND MOVE TO HOST MEMORY
;
        CALL    DPB$AD  ;GET DPB ADDR IN DE
        LXI     H,DD$DPB        ;DPB ADDR
        LXI     B,DPB$SZ        ;DPB SIZE IN BYTES
        CALL    BLOCK           ;MOVE INTO DPB
        LXI     H,DD$DPB        ;ID DTA DNS
        MOV     A,M     ;LOAD SEC/TRK
        CPI     26      ;TEST FOR SD
        CZ      TR3740  ;IF 0 USE 3740 TRN
        LHLD    DT$PTR  ;GET POINTER
        RET             ;EXIT BIOS
;
;( SET 3740 SECTOR TRANSLATION )*
;
TR3740: LXI     D,TRANS ;SECTOR TRAN TBL ADDR
        LHLD    DT$PTR  ;ADDR DISK PARA HDER
        MOV     M,E     ;SET UP TRANSLATE VECTOR
        INX     H
        MOV     M,D
        LXI     D,11    ;OFFSET TO DPB ADDR
        DAD     D       ;POINT TO DPB
        LXI     D,DPBSD0
        MOV     M,E     ;SET DPB TO SINGL DENS
        INX     H
        MOV     M,D
        XRA     A       ;A = 0
        STA     DNS     ;SAV IT
        RET             ;EXIT BIOS

;( GET DRIVE PARA BLK ADDR )

DPB$AD: LHLD    DT$PTR  ;ADDR DISK PARA HDER
        LXI     D,12    ;DPB TBL PNTR OFFSET
        DAD     D       ;NOW AT DPB PNTR
        MOV     E,M     ;LOW ORDER ADDR
        INX     H       ;NEXT BYTE
        MOV     D,M     ;HIGH ORDER ADDR
        RET     ;RETURN TO LOG USER
;
;
MOVE:   XCHG
```

only one bank using the rest of the memory for a RAMDISK. I went back and built several versions of the system with as much code in common as possible, and saved them for benchmarks.

My benchmarks are very oriented to the type of work that I do, like editing assembly language files, and assembling them. The source file I used to the test happens to be a format program which is relatively large (28K). The tree tests I used for comparison are the assembly of the file using MAC, read in PMATE and the file for editing, and lastly typing the file to the console. I always used the same diskette, and just changed the operating system on it. The six operating systems used are CP/M V2.2; CP/M V3.0 non-banked (NB); CP/M V3.0 with 1 bank (1B); CP/M V3.0 with 3 32K banks (3B); CP/M V3.0 with 1 bank and a RAM disk using the floppy drive (RD-A); and the same system as RD-A only using the RAM disk (RD-C). The computer is an S-100 with a 4 MHz. Z80 (JADE BIG Z) and a JADE DD Disk controller running 1 KB sectors on 8″ diskettes.

The above figures were obtained in only one run of each system and relied on the accuracy of a human and a stop watch. The RAM disk figures were not obtained using the same exact floppy, of course.

As you can see there isn't a great deal of change between systems. These tests are relatively unfair to the new features of CPM 3.0 since the assembly is CPU bound, and the TYPE is bound by the 9600 baud serial link to the terminal, but nonetheless, they are things I do.

The assembly time is best on the RAM disk, as expected. The buffering in S3 does help, but maybe the extra RAM is best used by making a RAM disk. CP/M 3 does not allow any control of the LRU (least record used) buffering algorithm, and therefore you are not able to optimize it for your application.

The loading of a file to edit is very dependent upon I/O. The banked systems seem to add about 30% to the I/O bound operation, and the RAM disk takes about 20% less. The files had not been used previously, so were not in the Data Buffers. When editing the file twice in a row, the second load also took 12.8 on S3. No improvement even though I allocated a 48 sector buffer. I went back and reGENed the system to have 240 1K buffers, and still no improvement. Maybe the LRU buffering is not working in my system.

The TYPE test was included since I thought that things were going slower, and they are, about 10%. The RAM disk makes up for the 10% lost by banking. The increased time to type a file in a banked system could be due to BDOS always saving the old stack, changing banks, and restoring the stack on a per charcter basis.

To do or not to do CP/M 3, that is the question. Well if you are looking for the added goodies like I/O redirection, ease in adding multiple disk controllers, editing command lines, fast warm boot, etc., then it is worth the effort. If you are looking for increased operating speed, stay with 2.2 and get a RAM disk. The things I really like are: fast warm boot, ease of installing RAM disk, confirm of wild card ERA, PIP, auto disk relog, I/O redirection, editing of previous command line, and the SAVE features.

I put CCP.COM and CPMLDR.COM on the system tracks, and while reading CCP during cold boot, save an image of it in bank 0. This gives me the feature of not having to have a system disk in drive A all the time. Due to auto density select, I still must use CNTRL-C if the floppies are of different densities. This makes a two drive system very easy to use.

However, I had several reservations about CP/M 3.

On encountering a disk error, CP/M3 returns to the CCP, whereas 2.2 would ignore the error if a return was typed, and continue what it was doing.

Will all my software run under 3.0? -Yes and No. I have not found anything that would not run in a nonbanked system on single density diskettes.

In a banked environment anything that jumps directly to the disk I/O BIOS vectors will not work. The disk routines are in bank 0, and the program is executing in bank 1. Since the wrong bank is in context when the routine is called, your system will probably jump to the right place in the wrong bank, and get lost. You should be able to correct this by changing the call to a BDOS DIRECT BIOS CALL.

If CP/M 3 is doing blocking and deblocking for the BIOS, any utility that calls the disk BIOS vectors directly will not work. In version 3 the entire sector is transferred on a read or write, instead of only the deblocked 128 byte segment as in V2.2. Also the maximum sector number for each track in CP/M 3 is the Physical number not the Logical. For example if you have 8 1KB sectors per track on your diskette, in V2.2 all calls to the BIOS are relative to the maximum sector number of 64 (eight physical sectors × eight 128-byte logical sectors per physical sector) and transfer 128 bytes per READ or WRITE. In version 3 the maximum sector number would be eight and every time you call the READ or WRITE, it puts 1024 bytes into your buffer. Many of the utilities like DU expect only 128 bytes, and therefore do not function properly.

There are reasons to go to CP/M 3, but performance increases on a floppy based system doesn't seem to be one of them. There are many nice features, but are they worth the $270 - 350 cost? Many of the features like hashed directories would be more beneficial on hard disk systems.

LISTING #1

```
        DB      0EDH,0B0H
        XCHG
        RET
;
;
HOME:   LXI     B,0         ;SET TRACK TO 0

        ; SETTRK
        ;       SET TRACK. SAVES TRACK ADDRESS FROM <BC>
        ;               IN @TRK FOR FURTHER OPERATIONS.

SETTRK:
        MOV L,C
        MOV H,B
        SHLD @TRK
        RET

        ; SETSEC
        ;       SET SECTOR.  SAVES SECTOR NUMBER FROM <BC>
        ;               IN @SECT FOR FURTHER OPERATIONS.

SETSEC:
        MOV L,C
        MOV H,B
        SHLD @SECT
        RET

        ; SETDMA
        ;       SET DISK MEMORY ADDRESS.  SAVES DMA ADDRESS
        ;               FROM <BC> IN @DMA AND SETS @DBNK TO @CBNK
        ;               SO THAT FURTHER DISK OPERATIONS TAKE PLACE
        ;               IN CURRENT BANK.

SETDMA:
        MOV L,C
        MOV H,B
        SHLD @DMA

        ; SECTRN
        ;       SECTOR TRANSLATE.  INDEXES SKEW TABLE IN <DE>
        ;               WITH SECTOR IN <BC>.  RETURNS PHYSICAL
        ;               SECTOR IN <HL>.  IF NO SKEW TABLE
        ;               (<DE>=0) THEN RETURNS PHYSICAL=LOGICAL.

SECTRN:
        MOV L,C
        MOV H,B
        MOV A,D
        ORA E
        RZ
        XCHG
        DAD B
        MOV L,M
        MVI H,0
        RET

; DISK I/O ROUTINES FOR STANDARDIZED BIOS INTERFACE

; INITIALIZATION ENTRY POINT.

;       CALLED FOR FIRST TIME INITIALIZATION.

INIT:
        XRA     A           ;SET FOR DRIVE ZERO
        STA     @RDRV
        RET
;
;
; READ A DISK SECTOR ROUTINE
;
READ:   LXI     H,READ$MSG
        SHLD    OPERATION$NAME
```

```
        CALL    BNK0    ;SWITCH TO BANK 0
        CALL    PMOVE   ;SET UP COMMAND BLOCK
        MVI     A,DC$RDS        ;READ SECTOR COMMAND
        CALL    DSK$EX          ;PERFORM OPERATION
        JNZ     DSK$ER          ;ERROR EXIT
        LHLD    @DMA            ;LOAD USER BUF ADDRESS
        XCHG                    ;MOVE HL TO DE
        LDA     DNS             ;GET DENSITY FLAG
        ORA     A
        CNZ     SETDD           ;NOT 0 MST BE DOUBLE
        LHLD    SEC$SZ          ;LOAD SECTOR SIZE
        MOV     B,H
        MOV     C,L     ;MOVE COUNT TO B
        LHLD    DD$BUF  ;LOAD BUFFER
        CALL    BLOCK           ;BLOCK MOVE ROUTINE
        JMP     DSK$OK          ;NORMAL RETURN
;
;
; DISK READ/WRITE EXITS
;
DSK$OK: CALL    BNK1    ;SWITCH DD TO BANK 1
        XRA     A       ;CLEAR A FOR GOOD END
        RET
;
;
;MOVE TRANSFER PARAMETERS TO DCM BLOCK
PMOVE:  LDA     @RDRV   ;GET DRIVE NUMBER
        STA     DISK    ;PUT IN COMMAND BLOC
        LHLD    @TRK
        MOV     A,L     ;GET LOW 8 BITS
        STA     TRACK
        LHLD    @SECT
        MOV     A,L
        STA     SECTOR
        RET
;
; DOUBLE D EXECUTION SUBROUTINE
;
;( COMMAND BLOCK TO DOUBLE D AND EXEC )
DSK$EX: STA     BT$CMD          ;STORE DCM COMMAND
        LXI     B,7             ;NMBR BYTE TO MOVE
        LXI     D,DD$CBT        ;COMMAND BYTE OFFSET
        LXI     H,BT$CMD        ;BIOS CMND BLOCK
        CALL    BLOCK           ;PERFORM BLOCK MOVE
        MVI     A,DC$INT        ;LOAD DD INTERRUPT
        OUT     D$PORT          ;ISSUE DD INTERRUPT

;( WAIT FOR DOUBLE D HALT )*

        MVI     B,01H   ;HALT MASK IN B REGISTER
DSK$WT: IN      D$PORT  ;READ DD STATUS
        ANA     B       ;TEST HALT* FLAG
        JNZ     DSK$WT  ;TEST UNTIL HALTED

;( GET DOUBLE D STATUS )

        CALL    BNK0    ;SWITCH DD ON TO BANK 0
        LDAX    D       ;GET STATUS BYTE
        MOV     M,A     ;STORE IN CMD BLK
        STA     ERFLAG
        ANA     A       ;TEST FOR ERRORS
        RET             ;RETURN TO CALLER
;
;TURN DD TO BANK 0
;
BNK0    MVI     A,DC$MB0        ;BANK 0 NUMBER
        OUT     D$PORT          ;SEND IT
        RET
;
;TURN DD TO BANK 1
;
BNK1    MVI     A,DC$MB1        ;BANK 1 VALUE
        OUT     D$PORT
        RET
;

; SUPPRESS ERROR MESSAGE IF BDOS IS RETURNING ERRORS TO
; APPLICATION...

DSK$ER: CALL    BNK1    ;SWITCH DD TO BANK 1
;
        LHLD    OPERATION$NAME
        CALL    PMSG            ; LAST FUNCTION

; THEN, MESSAGES FOR ALL INDICATED ERROR BITS
```

```
        LDA     BT$STS          ; GET STATUS BYTE FROM LAST ERROR
        LXI     H,ERROR$TABLE   ; POINT AT TABLE OF MESSAGE ADDR
ERRM1:
        MOV     E,M
        INX     H
        MOV     D,M
        INX     H       ; GET NEXT MESSAGE ADDRESS
        ADD     A
        PUSH    PSW     ; SHIFT LEFT AND PUSH RESIDUAL BITS WITH
STATUS
        XCHG
        CC      PMSG
        XCHG    ; PRINT MESSAGE, SAVING TABLE POINTER
        POP     PSW
        JNZ     ERRM1   ; IF ANY MORE BITS LEFT, CONTINUE

        LXI     H,ERROR$MSG
        CALL    PMSG            ; PRINT "<BEL>, RETRY (Y/N) ? "
        CALL    U$CONIN$ECHO    ; GET OPERATOR RESPONSE
        CPI     'Y'
        JZ      MORE$RETRIES    ; YES, THEN RETRY 10 MORE TIMES
HARD$ERROR:                     ; OTHERWISE,
        JMP     MONIT
;
MORE$RETRIES:
        LDA     BT$CMD
        ANA     A       ;WAS IT A LOGIN
        JZ      LOGIN
        JMP     READ
;
U$CONIN$ECHO:   ; GET CONSOLE INPUT, ECHO IT, AND SHIFT TO UPPER
CASE
        CALL    ?CONIN
        PUSH    PSW
        MOV     C,A
        CALL    ?CONO
        POP     PSW
        CPI     'A'
        RC
        SUI     'A'-'A'         ; MAKE UPPER CASE
        RET


        ; ERROR MESSAGE COMPONENTS

LOG$MSG:        DB      ' LOG-',0
READ$MSG        DB      ' READ-',0
OPERATION$NAME  DW      READ$MSG

        ; TABLE OF POINTERS TO ERROR MESSAGE STRINGS
        ;       FIRST ENTRY IS FOR BIT 7 OF 1797 STATUS BYTE

ERROR$TABLE     DW      B7$MSG
                DW      B6$MSG
                DW      B5$MSG
                DW      B4$MSG
                DW      B3$MSG
                DW      B2$MSG
                DW      B1$MSG
                DW      B0$MSG

B7$MSG          DB      ' NOT READY,',0
B6$MSG          DB      ' PROTECT,',0
B5$MSG          DB      ' FAULT,',0
B4$MSG          DB      ' RECORD NOT FOUND,',0
B3$MSG          DB      ' CRC,',0
B2$MSG          DB      ' LOST DATA,',0
B1$MSG          DB      ' DREQ,',0
B0$MSG          DB      ' BUSY,',0

ERROR$MSG       DB      ' RETRY (Y/N) ? ',0
;
; ID LABEL DEFINITIONS
;
JADEID: DB      'JADE'          ;ID LABEL
ID$SZE  EQU     $-JADEID        ;LABEL SIZE
;
; DOUBLE D - DCM COMMAND BLOCK BUFFER
;
BT$CMD: DB      0       ;DCM COMMAND
DISK:   DB      0       ;DRIVE NUMBER
TRACK:  DB      0       ;TRACK NUMBER
SECTOR: DB      0       ;SECTOR NUMBER
BT$SP0: DB      0       ;SPARE BYTE 0
```

```
BT$CHR: DB      0          ;LIST CHARACTER
BT$MOD: DB      0          ;MODE CONTROLS
BT$STS: DB      0          ;COMMAND STATUS
;
; BIOS VARIABLE STORAGE
;
DT$PTR: DW      FDSD0      ;DPH POINTER
DD$BUF: DW      DD$SBUF    ;SINGLE DENSITY BUFFER
SEC$SZ: DW      80H        ;STANADRD SECTOR SIZE
LOG$RQ: DB      0          ;LOG ON REQUEST REG
DNS:    DB      1          ;DENSITY 1 = DD; 0 = SD
ERFLAG: DS      1
;
    ; DISK COMMUNICATION DATA ITEMS

@ADRV   DS      1                    ; CURRENTLY SELECTED DISK DRIVE
@RDRV   DS      1                    ; CONTROLLER RELATIVE DISK DRIVE
@TRK    DS      2                    ; CURRENT TRACK NUMBER
@SECT   DS      2                    ; CURRENT SECTOR NUMBER
@DMA    DS      2                    ; CURRENT DMA ADDRESS

FDSD0   DW      TRANS
        DB      0,0,0,0,0,0,0,0,0
        DB      0          ;MEDIA FLAG
        DW      DPBSD
        DW      CVEC       ;CHECKSUM VECTOR
        DW      AVEC       ;ALLOCATION VECTOR
        DW      DIRBCB     ;DIRECTORY BUFFER CONTROL BLK
        DW      DTABCB     ;DATA BCB
        DW      0FFFFH     ;NO HASH
        DB      0          ;HASH BANK
;
CVEC:   DS      32
AVEC:   DS      40
;
DIRBCB: DB      0FFH       ;DRIVE NUMBER
        DB      0,0,0      ;RECORD #
        DB      0          ;WFLG
        DB      0          ;0
        DW      0          ;TRACK
        DW      0          ;SECTOR
        DW      BUF0       ;BUFFERADDR
        DB      0          ;BANK
        DW      0          ;LINK

DTABCB: DB      0FFH       ;DRIVE NUMBER
        DB      0,0,0      ;RECORD #
        DB      0          ;WFLG
        DB      0          ;0
        DW      0          ;TRACK
        DW      0          ;SECTOR
        DW      BUF1       ;BUFFERADDR
        DB      0          ;BANK
        DW      0          ;LINK
DPBSD   DPB     1024,9,77,2048,128,2     ;DOUBLE DENSITY
DPBSD0  DPB     128,26,77,1024,64,2      ;SINGLE DENSITY

TRANS   SKEW    26,6,1                   ;SINGLE DENSITY
TRANS1  DB      1,2,3,4,5,6,7,8,9        ;DOUBLE DENSITY

BUF0:   EQU     $
BUF1:   EQU     $+1024

        END
```

LISTING #2

```
;SYSTEM TEST PROGRAM
;LAST EDITED 08/10/83 16:13:46

SECSZ   EQU     80H
;
        ORG     2*SECSZ
        DB      'SECTOR 1 TRACK 0'
        ORG     3*SECSZ
        DB      'SECTOR 2 TRACK 0'
        ORG     4*SECSZ
        DB      'SECTOR 3 TRACK 0'
        ORG     5*SECSZ
        DB      'SECTOR 4 TRACK 0'
        ORG     6*SECSZ
        DB      'SECTOR 5 TRACK 0'
        ORG     7*SECSZ
        DB      'SECTOR 6 TRACK 0'
        ORG     8*SECSZ
        DB      'SECTOR 7 TRACK 0'
        ORG     9*SECSZ
        DB      'SECTOR 8 TRACK 0'
        ORG     10*SECSZ
        DB      'SECTOR 9 TRACK 0'
        ORG     11*SECSZ
        DB      'SECTOR 10 TRACK 0'
        ORG     12*SECSZ
        DB      'SECTOR 11 TRACK 0'
        ORG     13*SECSZ
        DB      'SECTOR 12 TRACK 0'
        ORG     14*SECSZ
        DB      'SECTOR 13 TRACK 0'
        ORG     15*SECSZ
        DB      'SECTOR 14 TRACK 0'
        ORG     16*SECSZ
```

```
DB      'SECTOR 15 TRACK 0'
ORG     17*SECSZ
DB      'SECTOR 16 TRACK 0'
ORG     18*SECSZ
DB      'SECTOR 17 TRACK 0'
ORG     19*SECSZ
DB      'SECTOR 18 TRACK 0'
ORG     20*SECSZ
DB      'SECTOR 19 TRACK 0'
ORG     21*SECSZ
DB      'SECTOR 20 TRACK 0'
ORG     22*SECSZ
DB      'SECTOR 21 TRACK 0'
ORG     23*SECSZ
DB      'SECTOR 22 TRACK 0'
ORG     24*SECSZ
DB      'SECTOR 23 TRACK 0'
ORG     25*SECSZ
DB      'SECTOR 24 TRACK 0'
ORG     26*SECSZ
DB      'SECTOR 25 TRACK 0'
ORG     27*SECSZ
DB      'SECTOR 26 TRACK 0'
ORG     28*SECSZ
DB      'SECTOR 1 TRACK 1'
ORG     29*SECSZ
DB      'SECTOR 2 TRACK 1'
ORG     30*SECSZ
DB      'SECTOR 3 TRACK 1'
ORG     31*SECSZ
DB      'SECTOR 4 TRACK 1'
ORG     32*SECSZ
DB      'SECTOR 5 TRACK 1'
ORG     33*SECSZ
DB      'SECTOR 6 TRACK 1'
ORG     34*SECSZ
DB      'SECTOR 7 TRACK 1'
ORG     35*SECSZ
DB      'SECTOR 8 TRACK 1'
ORG     36*SECSZ
DB      'SECTOR 9 TRACK 1'
ORG     37*SECSZ
DB      'SECTOR 10 TRACK 1'
ORG     38*SECSZ
DB      'SECTOR 11 TRACK 1'
ORG     39*SECSZ
DB      'SECTOR 12 TRACK 1'
ORG     40*SECSZ
DB      'SECTOR 13 TRACK 1'
ORG     41*SECSZ
DB      'SECTOR 14 TRACK 1'
ORG     42*SECSZ
DB      'SECTOR 15 TRACK 1'
ORG     43*SECSZ
DB      'SECTOR 16 TRACK 1'
ORG     44*SECSZ
DB      'SECTOR 17 TRACK 1'
ORG     45*SECSZ
DB      'SECTOR 18 TRACK 1'
ORG     46*SECSZ
DB      'SECTOR 19 TRACK 1'
ORG     47*SECSZ
DB      'SECTOR 20 TRACK 1'
ORG     48*SECSZ
DB      'SECTOR 21 TRACK 1'
ORG     49*SECSZ
DB      'SECTOR 22 TRACK 1'
ORG     50*SECSZ
DB      'SECTOR 23 TRACK 1'
ORG     51*SECSZ
DB      'SECTOR 24 TRACK 1'
ORG     52*SECSZ
DB      'SECTOR 25 TRACK 1'
ORG     53*SECSZ
DB      'SECTOR 26 TRACK 1'
ORG     54*SECSZ
DB      'SECTOR 27 TRACK 1'
ORG     55*SECSZ
DB      'SECTOR 28 TRACK 1'
ORG     56*SECSZ
DB      'SECTOR 29 TRACK 1'
ORG     57*SECSZ
DB      'SECTOR 30 TRACK 1'
ORG     58*SECSZ
DB      'SECTOR 31 TRACK 1'
ORG     59*SECSZ
DB      'SECTOR 32 TRACK 1'
ORG     60*SECSZ
DB      'SECTOR 33 TRACK 1'
ORG     61*SECSZ
DB      'SECTOR 34 TRACK 1'
ORG     62*SECSZ
DB      'SECTOR 35 TRACK 1'
ORG     63*SECSZ
DB      'SECTOR 36 TRACK 1'
ORG     64*SECSZ
DB      'SECTOR 37 TRACK 1'
ORG     65*SECSZ
DB      'SECTOR 38 TRACK 1'
ORG     66*SECSZ
DB      'SECTOR 39 TRACK 1'
ORG     67*SECSZ
DB      'SECTOR 40 TRACK 1'
ORG     68*SECSZ
DB      'SECTOR 41 TRACK 1'
ORG     69*SECSZ
DB      'SECTOR 42 TRACK 1'
ORG     70*SECSZ
DB      'SECTOR 43 TRACK 1'
ORG     71*SECSZ
DB      'SECTOR 44 TRACK 1'
ORG     72*SECSZ
DB      'SECTOR 45 TRACK 1'
ORG     73*SECSZ
DB      'SECTOR 46 TRACK 1'
ORG     74*SECSZ
DB      'SECTOR 47 TRACK 1'
ORG     75*SECSZ
DB      'SECTOR 48 TRACK 1'
ORG     76*SECSZ
DB      'SECTOR 49 TRACK 1'
ORG     77*SECSZ
DB      'SECTOR 50 TRACK 1'
ORG     78*SECSZ
DB      'SECTOR 51 TRACK 1'
ORG     79*SECSZ
DB      'SECTOR 52 TRACK 1'
ORG     80*SECSZ
DB      'SECTOR 53 TRACK 1'
ORG     81*SECSZ
DB      'SECTOR 54 TRACK 1'
ORG     82*SECSZ
DB      'SECTOR 55 TRACK 1'
ORG     83*SECSZ
DB      'SECTOR 56 TRACK 1'
ORG     84*SECSZ
DB      'SECTOR 57 TRACK 1'
ORG     85*SECSZ
DB      'SECTOR 58 TRACK 1'
ORG     86*SECSZ
DB      'SECTOR 59 TRACK 1'
ORG     87*SECSZ
DB      'SECTOR 60 TRACK 1'
ORG     88*SECSZ
DB      'SECTOR 61 TRACK 1'
ORG     89*SECSZ
DB      'SECTOR 62 TRACK 1'
ORG     90*SECSZ
DB      'SECTOR 63 TRACK 1'
ORG     91*SECSZ
DB      'SECTOR 64 TRACK 1'

END
```

iscontinued
dBASE II
(MULTI-USER VERSION)

PC WEEK

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.

Extended Single-Density Disk Storage

by Willis E. Howard, III

With an 8″ single-density system incorporating the Tarbell SD controller at home, I have been more than interested in the techniques that Bob Lurie has been discussing lately in *Microsystems* (June and October 1983). Essentially, here was a method of doubling my CBIOS size and increasing the data storage per disk by 50%, and it required absolutely no hardware modifications.

Before starting out, I decided on two requirements which my implementation had to adhere to. These were:

1) Track 0, sector 1 must be loadable with my current ROM loader. This means a standard 128-byte sector.
2) The CBIOS must provide compatibility with the standard data track format so that programs can be bought, sold, and exchanged without bizarre data transfer techniques.

An important aspect to point out at the beginning is that system track usage (0,1) is almost independent of data track usage (2-76). No matter what anybody does to the system tracks, if the data tracks are in the standard format (26 sectors of 128 bytes each), then we can all exchange our data and programs. On the other hand, it is necessary to use a nonstandard format on the system tracks if we are going to expand the CBIOS to include the additional code for the two-sector data track format.

Thus, the entire procedure is partitioned into two quite separate and distinct parts. First, a method of formatting the system tracks and loading a larger CBIOS must be developed. This includes modifying the SYSGEN program. Second, the question of formatting data tracks and modifying the CBIOS must be addressed. The CBIOS modification gets its starting point from the program DEBLOCK.ASM, which is supplied by Digital Research with CP/M 2.2.

Space for the CBIOS

The reformatting of the system tracks is worth doing, even if the two-sector method is not used for the data tracks. I have always been disappointed in the small amount of space available for implementation of the CBIOS. Page 13 of the Alteration Guide shows that only 7 sectors are available. After all of the necesary tables and subroutines are written, no space is left for simple things like including error messages or implementation of the IOBYTE.

Reformatting the system tracks from 26 to 29 sectors per track gives an extra 6 sectors *just for the CBIOS*. These extra sectors can be used for all the things you always wanted to do, but never had the space.

Three other techniques also come to mind for increasing the CBIOS size. The first involves using the AUTOLOAD feature of the CCP to load the last part of the CBIOS from a disk file. This was discussed by Andrew Bender in the March 1983 issue of *Microsystems.* Although this technique certainly works, it is at the expense of at least 1K of disk data space (the lowest allocation group size). It is a high price for us, the SSSD users, to pay.

A second technique involves using the unallocated sectors on track 76. CP/M data starts on track 2, following the directory. Since the standard allocation is in groups of 1K for a standard 8″SSSD disk, a short calculation will show that there are a few extra sectors at the end of the final track. In fact, Bob Lurie has listed them in the March 1983 issue of *Microsystems* as sectors 18, 24, 4, 10, 16 and 22 for a skew of six.

One drawback for usage of these tracks is the requirement for a really special SYSGEN. In addition, there is not enough room in a one-sector cold loader to individually address these sectors. Thus the cold boot routine of the CBIOS may have to read them in. I would consider writing code to boot from these sectors only as a last resort.

The third method would involve formatting track 1 as two sectors of 2,432 bytes each. With 29 sectors on track 0, this would give an effective 66 logical sectors for CP/M or 15 extra sectors for the CBIOS. Although I haven't tried this method, it should involve minimal changes to a cold loader and would probably fit into one logical sector. For this method, a special SYSGEN program will also be needed.

The following steps illustrate how I have implemented the 29-sector/system track and 2-sector/data track technique. The steps allow testing to occur at each critical stage. This should minimize the number of system crashes and make debugging much easier.

Step 1: The FORMAT program

In order to use any of these techniques, the first step is to write a format program. To do this, the FD1771 data sheet from Western Digital will prove invaluable. For Tarbell SD users, it is provided in the manual, starting on page 7-2-1.

The Western Digital data sheet has a section on the proper method of formatting a standard 8″ SSSD disk with 26 sectors/track. I wanted a program that would format any track on my disk with the IBM 26- or 29-sector formats or the non-IBM 2-sector format.

On Table 1, there is a summary of the data sent for 2-, 26-, and 29-sector/track formats. For the 2- and 29-sector formatting, the entire Index Address Mark and preceeding gap are eliminated. In addition, the gap between each of the 29 sectors is decreased to the absolute minimum as stated in the 1771 data sheet. The gap for the 2-sector/track format was increased to 103. Bob Lurie has noted that this large gap is necessary to compensate for possible drive speed variations that could overwrite the sector ID address marks.

After setting up the parameters given in Table 1, a few test programs showed that the data for the 2,432-byte sectors could not be calculated and written at the same time for a 2 MHz Z80 system. Thus my format program first filled memory with all the data for one track and then wrote the data to disk.

After writing each track, four tests check for possible errors:

1) Verify that all data has been sent to the disk by checking the memory image pointer.
2) Read the disk status register to test for a WRITE TRACK error.
3) After the first track is written in any formatting operation, issue a READ TRACK command and count the number of bytes on the track. If not 5208 +/- 50, the disk speed is assumed to be out of specification.
4) Verify each sector on the track by issuing standard READ commands and checking for read errors.

Step 2: Test the unmodified system

After you have a FORMAT program, it is a good idea to test the ability of your system to read and write the 29-sector tracks before anything is modified. To do this, simply format the system tracks of a blank disk (play it safe) with 29 sectors each. Then use your regular SYSGEN procedure to write CP/M to the newly formatted disk. The last three sectors of each track will not be written, but the question is whether or not the spacing is too narrow for your system to handle.

Now take the new disk and try to cold boot from it. After you can do this successfully, you are well on your way to using a large CBIOS on your system.

Step 3: New CP/M size

With expanded data track storage, there is one unavoidable drawback: the larger sectors must be buffered in the CBIOS. Although this buffer does not take up real disk space, it will use real memory space. Consequently, the TPA size of your system must be decreased.

In addition, if your are running a standard 64K system, the boot program for 29 sectors/track can wrap around to address zero and overwrite itself. This means that the system size must be decreased before the cold loader is modified.

Use MOVCPM to create a CP/M system which is 3 to 4K smaller than your current system. 2.4K of that will go to buffering the 2,432-byte sectors, and the rest is needed for the additional code in the CBIOS. It is possible that your CBIOS will not require such a large reduction in TPA size, especially if you have optimized your code. For your initial implementation, however, give yourself plenty of working space.

Be sure that you have modified your version of MOVCPM as documented by Digital Research and discussed by Bob Lurie in *Microsystems* (May, 1983). This patch eliminates a CP/M directory write bug and affects the large sector blocking. You may not have needed to fix it before, but you will now.

The DDT patch consists of modifying MOVCPM with the following code:

```
ORG     1CD2H
NOP
NOP
LXI     H,0
END
```

After MOVCPM.COM is loaded by DDT, the above code can be added either with the DDT assemble command "A" or by reading in the assembled HEX file of the code. While saving, you may want to rename the modified MOVCPM so that it can be distinguished from the unmodified version.

The new MOVCPM should be used to generate the smaller version of CPMxx.COM. After saving the new CPM file, edit and assemble your CBIOS and the Cold Loader to reflect the new system size. At this point, you should have files CPMxx.COM, CBIOSxx.HEX and CLOADxx.HEX, where xx refers to the smaller CP/M size.

Follow the normal second-level system generation procedures to place the new system machine code on the system tracks of a disk. Proceed to the next step only after you can successfully boot with the new system. From now on, use only the smaller size CP/M system.

Step 4: Cold loader for 29 sectors/track

For those using a modified version of the Tarbell Coldstart Loader, there are two definitions which must be changed. The lines

```
SPT      EQU    26
     ;Number of sectors per track
NSECTS   EQU    51
     ;Sectors of CP/M
```

must be changed to the following:

```
SPT      EQU    29
     ;Number of sectors per track
NSECTS   EQU    SPT*2-1
     ;Sectors of CP/M
```

Then just reassemble the program in preparation for the SYSGEN.

If you don't use the Tarbell loader, the sector loading algorithm is quite simple:

1) Start loading from Track 0, Sector 2.
2) If a total of NSECTS sectors has been loaded, jump to the CBIOS entry point.
3) If sector SPT has been loaded from the current track, step to the next track, load sector 1 and continue at (2).
4) Load the next sector and continue at (2).

These modifications to the cold loader do not affect its size. Give the new loader a new name such as CLOAD29.ASM. Assemble the new loader to get the file CLOAD29.HEX. It will not be tested until the new SYSGEN program is generated.

Step 5: A modified SYSGEN program

A section of the SYSGEN program is shown in Listing 1, with the modifications already made for 29 sectors/system track. The values at MXSEC and SECLST must be patched into your version of SYSGEN.

I have found that by using the sector ordering as given in SECLST, the SYSGEN will proceed somewhat more rapidly on my system. This ordering is different from that of Bob Lurie and makes the patch a little longer. It may be easier to make a HEX patch than to use DDT to directly make the changes. Save the patched SYSGEN program under a new name, such as SYSGEN29.COM.

Step 6: Test the modified system

In Step 2, the initial test showed that narrow gap sectors (29/track) could be read and written on your system. In Step 3, a smaller system size CP/M was generated. Now the modifications to the Cold Loader and SYSGEN programs can be tested.

Following the standard second level system generation procedure, load all modules into memory. With DDT, the following dialog results:

```
A>DDT CPMxx.COM
DDT VERS 2.2
NEXT PC
2300 0100
-ICLOAD29.HEX
-R900
NEXT PC
2300 0000
-ICBIOSxx.HEX
-R3580
NEXT PC
2300 0000
*^C
A>
```

With CP/M now in memory, use the new version of SYSGEN29.COM to put it on the system tracks of the 29 sector/system track disk created in Step 2. The dialog is as follows:

```
A>SYSGEN29
SYSGEN29 V 2.0
Source drive name (or return to
  skip)
Destination drive name (or
  return to reboot)A
Destination on A, then type
  return
Function complete
Destination drive name (or
  return to reboot)

A>
```

Take the disk and try to cold boot from it. If there are any problems, they can only be from the new Cold Loader, CLOAD29, or the new SYSGEN program, SYSGEN29. Go over your changes until this step works.

At this point, your system will boot in CP/M with a CBIOS of up to 15 sectors, 6 more than the standard size. The CBIOS can now be modified to access disks formatted with 2 sectors/data track or for any other application you may have.

Step 7: Modifying the CBIOS

Since the CBIOS provides the interface between the disk controller and the BDOS, it is necessarily hardware dependent. My system uses the Tarbell SD controller and two Pertec FD500 8″ SSSD drives. For comparison with the original Tarbell CBIOS, this requires

```
DUAL SET TRUE
```

in the definitions section. Keep the fact in mind that hardware differences may require a modification of some of the details discussed below.

A good way to start the CBIOS modifications is to first merge the program DEBLOCK.ASM into your CBIOS. Although some changes must be made for the method described here, the DEBLOCK program does describe some necessary steps. Be sure to use the copy supplied on your CP/M disk. There are a few differences from the version listed in the Alteration Guide. It is especially important to rewrite the SELDSK, SETTRK and SETSEC routines so that they only record the desired information and perform no actual selection. That must be delayed until the READ and WRITE routines.

The deblocking algorithm described in DEBLOCK.ASM is not one that can be used by us. Thus the SMASK macro can be eliminated and the CP/M host disk constants replaced with the display shown in Listing 2. The constants SECMSK and SECSHF will not be used.

LISTING #1

```
;
;       The origin statement is given for reference,
;       in case DDT is used to examine SYSGEN.COM.
;
        ORG     0128H
;
;       Total number of system tracks to read/write.
TRACKS:
        DB      2H
;
;       Total number of sectors per system track.
;       Standard value is 26.  Now set for 29 to
;       get extended CBIOS.
MXSEC:
        DB      1DH
;
;       This is a list of the sectors, from 1 to MXSEC.
;       The ordering may be set to the skew for your
;       system.  It is now set for two.  There is no real
;       conversion of logical to physical sectors.  This
;       table just gives the ORDER in which sectors are
;       read and written.  Unused table space zero filled.
SECLST:
        DB      1H,3H,5H,7H,9H,0BH,0DH,0FH
        DB      11H,13H,15H,17H,19H,1BH,1DH
        DB      2H,4H,6H,8H,0AH,0CH,0EH,10H
        DB      12H,14H,16H,18H,1AH,1CH
        DB      0H,0H,0H,0H,0H,0H,0H,0H,0H
        DB      0H,0H,0H,0H,0H,0H,0H,0H,0H
        DB      0H,0H,0H,0H,0H,0H,0H,0H,0H
        DB      0H,0H,0H,0H,0H,0H,0H,0H
```

LISTING #2

```
BLKSIZ    EQU    2048             ;Group size
HSTSIZ    EQU    2432             ;Bytes/sector
HSTSPT    EQU    2                ;Sectors/track
HSTBLK    EQU    HSTSIZ/128       ;Blocks/sector
CPMSPT    EQU    HSTBLK*HSTSPT    ;CPM blocks/track
```

LISTING #3

```
RWOPER:
          XRA     A           ;Zero to accum
          STA     ERFLAG      ;No errors (yet)
          LDA     SEKSEC      ;Compute host sector
          CPI     19          ;/19 NEW CODE
          MVI     A,1         ;Sector number 1
          JM      RW4
          INR     A           ;or 2
RW4:
          STA     SEKHST      ;Host sector to seek
```

LISTING #4

```
MATCH:
          LDA     SEKSEC      ;Mask buffer number
          CPI     19          ;MOD 19  NEW CODE
          JM      MAT3
          SBI     19
MAT3:
          MOV     L,A         ;Ready to multiply
          MVI     H,0
          REPT    7
          DAD     H
          ENDM
```

LISTING #5

```
MVI     A,0FFH      ;Auto disk type select on boot
STA     BSET        ;Deselect B:
XRA     A           ;Select disk A:
CALL    SETLNG      ;Get sector length code
STA     DISKA       ;Save it
```

You should also completely remove all variables that begin with "una," as well as all references to these variables. This is because prereads are always required, whether or not an unallocated group is being written to.

The problem is that the group size (2,048 bytes) is not a multiple of the physical disk sector size (2,432 bytes). This causes physical disk sectors to contain data belonging to two (sometimes three) different allocation groups. Even if data will be written to an unallocated group, the entire physical sector must be pre-read in case the other groups in the sector contain allocated data.

In the body of DEBLOCK.ASM, it is necessary to change the sections which do the logical/physical sector conversions. The code which converts to a physical sector number is shown in Listing 3.

The code which maps logical sector numbers onto the current physical sector buffer is shown in Listing 4.

In forming the disk tables and writing the disk routines, the strategy will be the following:

1) If a standard format disk is in a drive, read and write with no blocking.
2) If an extended storage format disk is in a drive, read and write with blocking.
3) If tracks 0 or 1 are accessed, read and write with no blocking, even if the disk has an extended storage format.

The host disk constants and blocking information will apply only to the extended storage format.

Next, let us consider the disk table definitions. These are shown, together with disk parameter storage, on Table 2. At DPE0 and DPE1 are the disk parameter blocks for disks A and B, using the standard format as described by disk descriptor table DPB0. At DPE2 and DPE3 are the parameter blocks for disks A and B, using the 2432 bytes/sector and 2 sectors/track format.

Although four disk parameter blocks are defined, only two are in use at any time, depending on which kind of disk is in which drive. This selection process occurs at cold and warm boot for drive A and at the first disk select for drive B, following each cold and warm boot.

The selection code is initiated in the SETUP routine, the common exit code of the cold and warm boot routines. It uses the code shown in Listing 5.

BSET is first set to -1 in order to indicate that drive B has been deselected; that is, the system does not know what kind of disk is in drive B. The subrou-

LISTING #6

```
;
;       Translate sector
SECTRN:
        MOV  H,B     ;Logical sector in HL
        MOV  L,C
        MOV  A,D     ;If zero XLT, no translation
        ORA  E
        RZ
        DAD  D       ;Otherwise, translate
        MOV  L,M
        MVI  H,Ø     ;Single byte only
        RET
```

TABLE I
Track Formatting Information

2 Sector #	2 Sector Value	26 Sector #	26 Sector Value	29 Sector #	29 Sector Value	Meaning
#	Value	#	Value	#	Value	# denotes the number of times (in decimal) that the Value (in hex) must be written.
Ø	--	40	ØØ	Ø	--	46 bytes follow the physical index and precede the index address mark.
Ø	--	6	ØØ	Ø	--	The last 6 MUST be Ø.
Ø	--	1	FC	Ø	--	Writes the Index Address Mark. This is used only for the standard 26 sector format.
11	FF	26	FF	11	FF	Post index gap. The gap in front of any data field must be at least 17 bytes long, but 32 is nominal most of the time. Use smallest gap for the non-standard sectors.
						Repeat the following for the sector count per track.
6	ØØ	6	ØØ	6	ØØ	The last 6 bytes in front of a data field MUST be Ø.
1	FE	1	FE	1	FE	Address Mark. This indicates the start of an ID record which contains track and sector addresses.
1	T#	1	T#	1	T#	This byte is the track number (Ø to 4C for up to 77 tracks).
1	ØØ	1	ØØ	1	ØØ	A single zero is required here.
1	S#	1	S#	1	S#	This byte is the sector number which begins at 1 and ends at the sector count per track.

tine SETLNG performs the following functions:

1) Select the disk designated by A (drive A if 0).
2) Seek the third track (track 2).
3) Issue a read address command (0C4H) to the 1771 and get the sector length byte.
4) Return a 0 for a zero sector length, which indicates a standard format. Return a 2 for a nonzero sector length, indicating a 2432-byte/sector format.

The result is stored in DISKA. In case of a read error, there should be 10 retries before reporting an error.

The first time (after a cold or warm boot) that a call is made to SELDSK for the selection of drive B. SETLNG will be called and the result plus one will be stored in DISKB. An initial call is assumed whenever BSET is -1. The SELDSK routine must also be modified to use DISKA or DISKB as the value of SEKDSK and in the computation of the disk parameter block address.

Using this procedure, a disk with either format can be placed in either drive. If disk formats are changed, the control-C key should be immediately pressed while in the CP/M command mode. This will force the system to test the format type of the disks before reading and writing. Never change a disk if it contains open files.

Before moving on to the READ/WRITE routines, the sector translation routine must be considered. In this implementation, there are two types of formats: the standard format, which must undergo a sector translation with a skew of 6, and the extended storage format, which needs no sector translation. The code shown in Listing 6 can handle both formats.

The CBIOS READ/WRITE entry points must perform the following:

1) Clear the extended sector read/write flag.
2) If track 0 or 1 selected, do a standard read/write.
3) If an extended sector disk is selected, do read/write with blocking as described in DEBLOCK.ASM. If this requires a physical read/write, set the extended read/write flag and call the appropriate routine.
4) Otherwise, do a standard read/write, as in the unmodified CBIOS.

Where possible, make only a few changes at a time. Then do a SYSGEN and test the changes. This way, errors can be easily found and corrected.

User beware

Of course, there are dangers. When

1	98	1	ØØ	1	ØØ	Sector length. This is not a real length but a code which has different meanings for IBM and non-IBM formats. For IBM 128 byte sectors it is Ø and for non-IBM 2432 byte sectors, 98.
1	F7	1	F7	1	F7	This causes two CRC bytes to be written. This byte ends the ID record.
11	ØØ	11	ØØ	11	ØØ	First part of a 17 byte gap in front of the user data. It can also write FF.
6	ØØ	6	ØØ	6	ØØ	The last 6 bytes in the gap preceeding the user data field MUST be zero.
1	FB	1	FB	1	FB	Data address mark. This signals the start of the user data record.
2432	E5	128	E5	128	E5	Fill the user data record with E5. This is especially important for the directory sectors where E5 denotes an erased file.
1	F7	1	F7	1	F7	Write 2 checksum bytes (CRC's) to end the user data record.
1Ø3	FF	27	FF	14	FF	Setup the first part of the gap between records. This can have ØØ or FF data.
--						End single sector data. After all sectors have been written, fill the rest of the sector with FF until the 1771 interupts out.

TABLE II
Tables and parameter storage for Extended Storage CBIOS

```
;
;       DEFINE DISK PARAMETERS
DPBASE:                                 ;Base of parameter block
;       Standard sectors
DPEØ:
        DW      XLTØ,ØØØØH              ;Translate table
        DW      ØØØØH,ØØØØH             ;Scratch area
        DW      DIRBUF,DPBØ             ;Dir buff, param block
        DW      CSVØ,ALVØ               ;Check, alloc vectors
;
DPE1:
        DW      XLT1,ØØØØH
        DW      ØØØØH,ØØØØH
        DW      DIRBUF,DPB1
        DW      CSV1,ALV1
;
;       Large sectors
DPE2:
        DW      Ø,Ø
        DW      Ø,Ø
```

```
        DW      DIRBUF,DPB2
        DW      CSV2,ALV2
;
DPE3:
        DW      0,0
        DW      0,0
        DW      DIRBUF,DPB3
        DW      CSV3,ALV3
;
;       DISK PHYSICAL DESCRIPTION
;       Standard 8" SSSD
DPB0:
DPB1:
        DW      26              ;Sectors/track
        DB      3               ;Block shift
        DB      7               ;Block mask
        DB      0               ;Extnt mask
        DW      243-1           ;Disk size -1
        DW      64-1            ;Directory max
        DB      11000000B       ;Alloc0
        DB      00000000B       ;Alloc1
        DW      16              ;Check size
        DW      2               ;Number of sys tracks
;
;       Modified 8" SSSD w/ 2432 bytes/sector & 2 sectors/track
DPB2:
DPB3:
        DW      38              ;Logical sectors/track
        DB      4               ;BLS=2048
        DB      15
        DB      1               ;EXM
        DW      178-1           ;DSM
        DW      64-1            ;Directory
        DB      10000000B       ;Alloc
        DB      00000000B
        DW      16              ;Check size
        DW      2               ;System track count
;
;       Standard sector translation table
XLT0:
XLT1:
        DB      1,7,13,19,25
        DB      5,11,17,23,3
        DB      9,15,21,2,8
        DB      14,20,26,6,12
        DB      18,24,4,10,16,22
;
```

you start using a new system, especially one that gives you an extended storage capability, the tendency is to use it all the time. In fact, I used the system during the preparation of this article. The danger is that if your system goes down, many of your disks and programs can not be read by anyone you know or on any system you have access to. The moral is to keep primary backups on standard formatted disks.

Thus far, I have experienced no problems with the new system. Routine activities such as word processing, assembling, compiling and linking have given no errors. Even DUU worked like a champ. My CBIOS lets me mix standard and dual sector disks so well that I sometimes need to get STAT to find out which format a disk actually has.

For those interested in the application of this method, complete source listings are available for the FORMAT, Cold Loader, SYSGEN29 and CBIOS programs, as well as the MOVCPM patch. They have been submitted to the SIG/M library of public domain software.

```
;       Parameter storage
ALV0:
ALV2:
        DS      31      ;Allocation area
CSV0:
CSV2:
        DS      16
ALV1:
ALV3:
        DS      31
CSV1:
CSV3:
        DS      16
;
DIRBUF: DS      128
```

Assembly Language Extensions for Microsoft Basic

by Ron Kreymborg

Recently I was asked to write some graphics software for a Tektronix compatible terminal using Microsoft Basic. About the same time a number of schools requested programs, also in Basic 80, for experimental monitoring and control via the I/O facilities of their computers. Both tasks seemed ideal candidates for assembly language subroutines, particularly as they required fast execution with extensive bit handling. However, for school use a quick and easy method for modifying the routines was essential, and interfacing assembly routines to the Basic interpreter had always been a complex business at best. With this in mind, I began looking around for some ideas on loading and linking assembly modules to Basic.

Some additional requirements were added as the search progressed: the system must be easily portable between different CP/M computers, and a Basic program should be able to easily find the subroutine addresses at run time. The dreary business of fixed memory locations for subroutines was ruled out very early. This meant some form of automatic assembly language module relocation, as Basic assumes free memory begins immediately after its last address. For a system to be portable it would need to determine its environment at start up time the same as Basic, using the various memory boundries provided by CP/M.

Although the search included most of the usual sources, I could not find a method that provided exactly what I wanted. However, I did find an article on extensions to the CP/M BIOS by Mike Karas [1] in which he used a relocation technique previously described by Gary Kildall [2]. Here the assembler HEX file is processed to produce a Page Relocatable file (PRL), where addresses that require adjustment during relocation are flagged in a bit map appended to the file. Relocation consists of moving the file to the destination area and adjusting the contents of the flagged bytes to reflect their new location.

My approach

This started me thinking. Perhaps a small program using this technique could be permanently appended to Basic 80, arranged so it ran before Basic, and with a fixed task of supervising the relocation of an assembly language module to the top of available memory. If the assembly module itself could be repeatedly modified, extended, or otherwise changed but always loaded at same fixed address, the requirement for quick and easy modification would be satisfied.

The hardest part was arranging things so the Basic program could determine the number of subroutines and their addresses at run time. A little more brain storming and I had the solution: the first statement in the assembly language module is a jump followed by a byte containing the number of subroutines in the module, followed by the addresses of each subroutine. As an example, if there were two assembly routines called PLOT and UNPAK, the preamble to the module would look like:

```
bdos:  jmp   0       ; becomes BDOS address
       db    2       ; number of routines
       dw    plot    ; address of first
       dw    unpak   ; address of second
```

When the relocator program gets control, it moves the module to the highest possible address in memory, performs the bit map directed relocation, and extracts the CP/M BDOS address from memory and writes it to the jump instruction at the start of the relocated module. Finally it takes the module's new address and substitutes this for the old BDOS address.

A Basic program may now easily locate the start of the module by reading the address in the BDOS jump at location 0005H. By ensuring the relocation is always to a page boundry, only one byte need be read to find the module's address. Once this address is known, the number of subroutines in the module and their addresses may easily be found because of the standardized structure which forms a preamble to the module. Normal BDOS calls occur as usual with the addition of one extra jump. Moreover, when Basic 80 computes the size of memory available by reading the BDOS location, the module will be included in the calculation and is therefore protected from overwriting.

Although this method allows a Basic program to check if the currently attached module has at least the expected number of subroutines, there is no way to check if they are the right ones. If this is a problem, perhaps where a number of users are developing different modules, the <count> byte could be replaced with a revision or code number that must match a similar number in the Basic program. Perhaps the easiest method is to name YOUR Basic with a distinguishing title, like GRAFBAS.

The preamble standardization in the module is reflected in the Basic program, the first few lines of which are associated with finding out where the module is and what the various addresses are. I decided the CALL method for executing an external subroutine would be easier to use than the USR function, particularly as some users would eventually be compiling their finished programs. Continuing with the example above, the first few lines of a matching Basic program would be:

```
100 A = PEEK(7)*256          ' MODULE ADDRESS
110 A = A+3                  ' SKIP BDOS VECTOR
120 N = PEEK(A)
130 IF N <> 2 THEN PRINT "MODULE DOES NOT MATCH" : STOP
140 DEF FNA (I) = PEEK(I)+PEEK(I+1)*256-2^16
150 PLOT% = FNA (A+1)
160 UNPAK% = FNA (A+3)
```

To ensure the relocator runs first, the normal start address of Basic 80 is changed to point to the relocator, now inserted after the last address in Basic. The jump which exits the relocator now jumps to the old start address, and Basic then proceeds with its own initialization. Note that the space previously taken by the relocator and original module is completely recovered for data space as Basic 80 itself is quite unaware of the module's existence. Further, the relocator can check if a module is attached and move straight into the Basic initialization sequence if not, so the relocator may be left attached to Basic whether assembly modules are used or not.

Modifying the subroutine modules

Once the relocator is appended, adding or modifying subroutine modules is easy. The module is edited and perhaps debugged as a separate module if required, then assembled and linked using the submit file of Figure 1. This assumes there is no ORG statement in the module, and all references to the BDOS address are via the label called BDOS in the standardized preamble to the module (not to address 5). The two ORG files consist of a single line each. The ORG000 file contains:

```
ORG 000H
```

while the ORG100 file contains:

```
ORG 100H
```

The first two PIP statements in the submit file append copies of the module source code onto these ORG lines and the combined files are then assembled. The final PIP concatenates the two resulting hex files into one. The GENMOD program reads this file, writing the first half out again, then checking for bytes which do not match in the second half. These bytes should, of course, be the high bytes of all addresses which will require relocation at load time.

By relating each byte from the start of the file with a single bit, setting the bit whenever relocation is required, and appending the resulting bit map to the output file, GENMOD provides all the information a relocating program needs to position the file on any page boundry in memory. The GENMOD program is written in Basic and listed in figure 2. It is based on the article by Gary Kildall and provides all the necessary features for generating the relocatable module. The relocator module itself is very similar to that described by Mike Karas and is listed in Figure 3. The original relocator and modules were written in Z80 code for a Micromation system, and were therefore somewhat simpler. However this article was written for users with only the basic CP/M tools and Microsoft Basic.

The Basic I used was revision 5.2 and it ended at 6000H. This is therefore the start address of the RELOC program. If you have another version of Basic, check its ending address with DDT and use that in all that follows. After assembling the relocator of Figure 3, the DDT session required for its once only installation follows:

```
A>DDT MBASIC.COM
DDT VERS 1.1
NEXT PC
6000 0100
-IRELOC.HEX (read RELOC program
                         ORGed at 6000H)
-R
NEXT PC
6060 0000
```

The <stack> address is 6060H, and this is where the assembly module will be loaded to.

There are two changes to Basic that ensure it jumps to the relocator first, then jumps back after the relocation is complete. First change the jump at 100H:

```
-L100
0100 JMP 5D7E (In revision 5.2)
0103 MOV A,C
0104 DAD H
......
```

This must be changed to jump to the start address of RELOC.

```
-A100
0100 JMP 6003
0103
```

Now change the exit point in RELOC to the address previously at location 100H.

```
-L6000
6000 JMP 0000
6003 LXI SP,6060
......
-A6000
6000 JMP 5D7E
6003
```

Have a look at these changes to ensure there were no mistakes:

```
-L100
0100 JMP 6003
0103 MOV A,C
0104 DAD H
......
-L6000
6000 JMP 5D7E
6003 LXI SP,6060
......
```

Now exit DDT and save the modified Basic under some other name. Here we will use BASM for convenience.

```
-G0
A>SAVE 96 BASM.COM
```

An example

As an example of an assembly language module, the one in Figure 4 includes three routines: the first splits one integer into two, made up of both bytes in the original. The next two provide access to the discK directory, a feature normally unavailable in Microsoft Basic, but one rather nice to have for file searches etc. Figure 5 is a simple Basic program using these functions showing typical initialization and calling techniques. Remember that for up to three arguments, the ADDRESS of the first argument is passed in HL, the second in DE, and the third in BC.

The DDT session to append this module to our previously modified version of Basic follows. The offset required to read the module into memory so it starts at the <stack> address in the relocator may be conveniently computed using the DDT H command. Note that changing a module is very easy once this offset constant is known, and may be conveniently executed with a simple submit file.

```
A>DDT BASM.COM
DDT VERS 1.1
LAST PC
6100 0100
-H6060,100 (100H is normal CP/M start address)
6100 5F60 (use 5F60H difference as offset)
-IMODUL.PRL
-R5F60
NEXT PC
610E 0000
-G0
A>SAVE 97 BASM.COM
```

A typical submit file to automate this process is listed below. Note the number of blocks to save (97 here) would require modification for larger modules.

```
; Syntax is:
SUBMIT LINK <BASICNAME> <MODULENAME>

XSUB
DDT $1.COM
I$2.PRL
R5F60
G0
SAVE 98 $1.COM
```

I hope you will find these techniques handy for all those little jobs so easily handled in assembly language, but awkward in Basic. If you really need speed, algorithms like sorts are appreciably faster when written in assembler, and this technique provides for easy modification during program development. In addition, access to all BDOS calls (and XDOS calls in MP/M) are simple to implement.

Bibliography

1. A Dynamic CPM BIOS Extension Technique, M.Karas, LIFELINES, Volume 3, No 8.

2. A Simple Technique for Static Relocation Of Absolute Machine Code, G.Kildall, Digital Research.

3. BASIC-80 Reference Manual, Microsoft Inc.

Ron Kreymborg is a Computer Science graduate of Monash University. He has many years of electronics experience in high tech engineering fields, and over the last ten years has been increasingly involved in programming minis and micros in a number of languages for a variety of applications. He is currently with Microprocessor Applications, a microcomputer manufacturing firm in Melbourne, Australia.

FIGURE 1

This submit file generates a relocatable PRL version of the assembly language module. The Basic GENMOD program could be modified to pick up its input from the CPM command line tail area at 80H, making the whole process automatic. Note that in reality this organization produces a Digital Research SPR or RSP type file rather than a strict PRL file. I have kept the PRL name for convenience.

```
ERA $1.?
PIP TEMP.ASM=ORG000.ASM,$1.ASM
ASM TEMP
REN $1.0=TEMP.HEX
PIP TEMP.ASM=ORG100.ASM,$1.ASM
ASM TEMP
REN $1.1=TEMP.HEX
PIP $1.HEX=$1.0,$1.1
MBASIC GENMOD
ERA TEMP.*
ERA $1.?
```

FIGURE 2

The following Basic program provides the bare bones for a relocatable module generator. It assumes the input file will be of type HEX, while the output relocatable file is type PRL. In the interests of brevity and speed it provides only the minimum of error checking. It should perhaps check for valid hex characters and a correct checksum on each line.

```
1000 '                          GENMOD
1010 '
1020 '  A Microsoft Basic program which simulates the
1030 '  Digital Research program called GENMOD.COM.
1040 '  It takes a concatenated HEX file and produces
1050 '  a page relocatable version.
1060 '
1070 '                                                  Ron Kreymborg
1080 DEFINT A-Z
1090 INPUT "FILENAME (.HEX ASSUMED)? ",A$
1100 B$=A$+".HEX"
1110 S=6000
1120 DIM M(S), B(S/8)
1130 H$="0123456789ABCDEF"
1140 NOPRNT=1
1150 '
1160 OPEN "I",1,B$
1170 PASS=1
1180 FLG=0
1190 G=0
1200 PRINT
1210 PRINT "FIRST PASS"
1220 IF EOF(1) THEN 1280
1230 LINE INPUT #1, L$
1240 IF MID$(L$,2,2) = "00" THEN 1320
1250 GOSUB 2120
1260 GOTO 1220
1270 '
1280 CLOSE
1290 PRINT "INCORRECT FILE END"
1300 STOP
1310 '
1320 GOSUB 2370
1330 G=256
1340 PASS=2
1350 PRINT "SECOND PASS"
1360 IF EOF(1) THEN 1280
1370 LINE INPUT #1, L$
1380 IF MID$(L$,2,2) = "00" THEN 1420
1390 GOSUB 2120
1400 GOTO 1360
1410 '
1420 CLOSE
1430 Q=P+1
1440 FOR I=0 TO D
1450    P=P+1
1460    M(P)=B(I)
1470 NEXT I
1480 IF NOPRNT THEN 1690
1490 J=0
1500 PRINT
1510 PRINT "OUTPUT CODE:"
1520 PRINT
1530 T=512 : GOSUB 2480 : PRINT T$;"  ";
1540 FOR I=0 TO P
1550    IF J<16 THEN 1620
1560    PRINT "  ";
1570    FOR K=I-J TO I-1
1580       IF M(K)>=32 AND M(K)<=126 THEN PRINT CHR$(M(K)); ELSE PRINT ".";
1590    NEXT K
1600    PRINT
1610    J=0 : T=512+I : GOSUB 2480 : PRINT T$;"  ";
1620    T=M(I) : GOSUB 2480
1630    PRINT MID$(T$,3);" ";
1640    J=J+1
1650 NEXT I
1660 PRINT
1670 '
1680 ' NOW WRITE THE PRL FILE
1690 B$=A$+".PRL"
1700 A$="0000000000000000"
1710 OPEN "R",1,B$,16
1720 FIELD 1, 1 AS B0$, 2 AS B1$, 13 AS B2$
1730 LSET B0$=CHR$(0)
```

```
1740 FOR I=1 TO 14
1750    MID$(A$,I,1)=CHR$(0)
1760 NEXT I
1770 PH=Q\256
1780 PL=Q MOD 256
1790 B$=CHR$(PL)+CHR$(PH)
1800 LSET B1$=B$
1810 LSET B2$=A$
1820 PUT 1, 1
1830 LSET B1$=MKI$(0)
1840 FOR I=2 TO 16
1850    PUT 1, I
1860 NEXT I
1870 FIELD 1, 16 AS BUF$
1880 N=P\16+1
1890 P=0
1900 FOR I=1 TO N
1910    FOR J=1 TO 16
1920       MID$(A$,J,1)=CHR$(M(P))
1930       P=P+1
1940    NEXT J
1950    LSET BUF$=A$
1960    PUT 1, I+16
1970 NEXT I
1980 CLOSE
1990 PRINT
2000 T=Q : GOSUB 2480 : PRINT T$;" TOTAL CODE"
2010 T=P-1 : GOSUB 2480 : PRINT T$;" TOTAL BYTES"
2020 IF FLG THEN 2040
2030 PRINT "MODULE CONSTRUCTED" : GOTO 2050
2040 PRINT "ERRORS IN RE-LOCATION"
2050 PRINT
2060 SYSTEM
2070 END
2080 '
2090 '
2100 '
2110 ' TRANSLATE THE CURRENT HEX LINE
2120 T$=MID$(L$,2,2) : GOSUB 2440
2130 N=T
2140 T$=MID$(L$,4,2) : GOSUB 2440
2150 P=T*256
2160 T$=MID$(L$,6,2) : GOSUB 2440
2170 P=P+T-G-1
2180 J=10
2190 FOR I=1 TO N
2200    P=P+1
2210    T$=MID$(L$,J,2) : GOSUB 2440
2220    IF PASS = 1 THEN M(P)=T : GOTO 2320
2230    IF T = M(P) THEN 2320
2240    T=T-M(P)
2250    IF (T = 1 OR T = -255) THEN 2290
2260    PRINT HEX$(P);"RE-LOCATION ERROR";T
2270    FLG=1
2280    GOTO 2320
2290    P1=P\8
2300    P2=P MOD 8
2310    B(P1)=B(P1)+2^(7-P2)
2320    J=J+2
2330 NEXT I
2340 RETURN
2350 '
2360 ' ZERO THE BIT MAP
2370 D=P\8+1
2380 FOR I=0 TO D
2390    B(I)=0
2400 NEXT I
2410 RETURN
2420 '
2430 ' HEX TO DECIMAL
2440 T=(INSTR(H$,LEFT$(T$,1))-1)*16+INSTR(H$,RIGHT$(T$,1))-1
2450 RETURN
2460 '
2470 ' DECIMAL TO HEX
2480 T$="000"+HEX$(T)
2490 IF LEN(T$)>4 THEN T$=MID$(T$,2) : GOTO 2490
2500 RETURN
```

FIGURE 3

This is the relocator. It resides on top of Basic at location 6000H. It is assembled using the CPM ASM assembler and the HEX output file is used directly.

```
;*****************************************************************;
; RELOC:   A program to move a page relocatable assembly          ;
;       language module with an origin at zero to the top         ;
;       of available memory.                                      ;
;                                                                 ;
;                                           Ron Kreymborg         ;
;                                                                 ;
; This code resides at the end of MBASIC                          ;
;*****************************************************************;

        org     6000h

exit:   jmp     0               ; original address of Basic

start:  lxi     sp,stack

; Define code length and start address

        lxi     h,buff          ; source for move
        mov     c,m             ; get length of module
        inx     h
        mov     b,m
        mov     a,b             ; anything there?
        ora     c
        jz      exit            ; ..no

        lxi     d,0feh
        dad     d               ; start of code
        xchg                    ; put in DE
        push    b               ; save count
        push    b

; Compute destination address in HL

        lhld    6               ; get bdos address
        shld    buff+100h       ; insert in module
        mvi     1,0             ; ensure we go to a page boundry
        inr     b               ; convert length to pages
        mov     a,h
        sub     b
        mov     h,a

; Now move the code to the address in HL

        pop     b               ; restore byte count
        push    h               ; save destination address
movlp:  ldax    d               ; get byte of code
        mov     m,a             ; move to new location
        inx     h               ; step pointers
        inx     d
        dcx     b               ; and decrement counter
        mov     a,c
        ora     b               ; done?
        jnz     movlp           ; ..no, loop

        pop     h               ; restore destination
        shld    6               ; setup new bdos vector
        xchg
        pop     b               ; restore count

; Now BC has the byte count, HL points to the first byte of the
; bit map, and DE has the destination address

        push    h               ; bit map address to top of stack
        mov     h,d             ; (h) is relocated page offset

reclp:  mov     a,b             ; check if done
        ora     c
        jz      exit            ; ..yes, go start up Basic
        dcx     b               ; decrease count
        mov     a,e             ; is DE address modulo 8 bytes?
        ani     7               ;   if so - must get next map byte
        jnz     rec2            ; ..no

; Get next map byte via top of stack

        xthl                    ; get current map pointer
        mov     a,m             ; get byte into A
        inx     h               ; point to next byte
        xthl
        mov     l,a             ; store in L

rec2:   mov     a,l             ; check next bit in byte
        ral
        mov     l,a             ; save for later
        jnc     rec3            ; no relocation required

; Add in the offset for a set bit

        ldax    d               ; get byte
        add     h               ; add offset
        stax    d               ; store relocated byte back
rec3:   inx     d               ; step to next byte in relocated
        jmp     reclp           ;   code and around again

; <stack> must occur on an 8 bit boundry

        org     ($ AND 0fff0h) + 10h

stack   equ     $
        db      0
buff:   dw      0       ; zero for no module case

        end
```

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

Z-80 Macroassembler

Power for larger programs! This 2500AD macroassembler includes:

- Zilog Z-80 Macroassembler (with the same powerful features as all our assemblers)
- powerful linker that will link up to 128 files. Com files may start at any address
- Intel 8080 to Zilog Z-80 Source Code Converter (to convert all your Intel source to Zilog Syntax in one simple step)
- COM to Hex Converter (to convert your object files to Hex for PROM creation, etc.)
- 52 page User Manual

8086/88 Assembler with Translator

Available for MSDOS, PCDOS, or CPM/86! This fully relocatable macroassembler will assemble and link code for MSDOS (PCDOS) AND CPM/86 on either a CPM/86 or MSDOS machine. This package also includes:

- An 8080 to 8086 source code translator (no limit on program size to translate)
- A Z-80 to 8086 translator
- 64 page user manual
- 4 linkers included:
  - MSDOS produces .EXE file
  - CPM/86 produces .CMD file
  - Pure object code generation
  - Object code and address information only

Linker features:

- Links up to 128 files
- Submit mode invocation
- Code, Data Stack and extra segments
- Handles complex overlays
- Written in assembly language for fast assemblies.

Z-8000 Cross Development Package

Instant Z-8000 Software! This package allows development and conversion of software for the Z8001, 8002, 8003 and 8004 based machines on a Z-80, Z-8000 or 8086 machine. This powerful package includes:

- a Z-80/8080 to Z-8000 Assembly Language Source Code Translator
- Z-8000 Macro Cross Assembler and Linker

The Translators provide Z-8000 source code from Intel 8080 or Zilog Z-80 source code. The Z-8000 source code used by these packages are the unique 2500AD syntax using Zilog mnemonics, designed to make the transition from Z-80 code writing to Z-8000 easy.

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

Relocatable Code — the packages include a versatile Linker that will link up to 128 files together, or just be used for external reference resolution. Supports separate Code and Data space. The Linker allows Submit Mode or Command Invocation.

Large File Handling Capacity —the Assembler will process files as large as the disk storage device. All buffers including the symbol table buffer overflow to disk.

Powerful Macro Section— handles string comparisons during parameter substitutions. Recursion and nesting limited only by the amount of disk storage available.

Conditional Assembly—allows up to 248 levels of nesting.

Assembly Time Calculator— will perform calculations with up to 16 pending operands, using 16 or 32 Bit arithmetic (32 Bit only for 16 Bit products). The algebraic hierarchy may be changed through the use of parentheses.

Include files supported—

Listing Control—allows listing of sections on the program with convenient assembly error detection overrides, along with assembly run time commands that may be used to dynamically change the listing mode during assembly.

Hex File Converter, included —for those who have special requirements, and need to generate object code in this format.

Cross reference table generated—

Plain English Error Messages—

System requirements for all programs: Z-80 CP/M 2.2 System with 54k TPA and at least a 96 column printer is recommended. Or 8086/88 256k CP/M-86 or MSDOS (PCDOS).

Cross Assembler Special Features

Z-8—User defined registers names, standard Zilog *and* Z-80 style support. Tec Hex output option.

8748—standard Intel *and* Z-80 style syntax supported.

8051—512 User defined register or addressable bit names.

6800 Family—absolute or relocatable modes, all addressing modes supported. Motorola syntax compatible. Intel Hex or S-Record format output.

6502—Standard syntax or Z-80 type syntax supported, all addressing modes supported.

FIGURE 4

This is the example relocatable assembly language module for Basic. It demonstrates the standard preamble and a number of simple subroutines.

```
;*****************************************************************;
; Sample module.  Preamble begins..                               ;
;*****************************************************************;

bdos:   jmp     0               ; bdos vector
        db      (j-$)/2         ; routine count

        dw      split           ; split an integer
        dw      getlst          ; BDOS call 17
        dw      getnxt          ; BDOS call 18
        dw      setdma          ; BDOS call 26

j       equ     $

;*****************************************************************
; Split an integer into its seperate bytes.
;       Basic call syntax is:
;               CALL SPLIT% (SORCE%, LBYT%, RBYT%)

split:  mov     a,m             ; get right byte
        push    psw             ;   and save it

        inx     h               ; get left byte
        mov     a,m
        xchg                    ; and put in LBYT
        mov     m,a
        inx     h
        xra     a
        mov     m,a

        mov     h,b             ; right byte pointer into HL
        mov     l,c
        pop     psw             ; restore right byte
        mov     m,a             ;   and put in RBYT
        inx     h
        xra     a
        mov     m,a
        ret

;*****************************************************************
; BDOS call 17 - search for first matching entry in the disc
; directory. Must be preceded by a set dma call. Call with
; the syntax:
;       CALL GET1ST% (FCB%(0), PTR%)
; Returns:
;       PTR = 0 thru 3  ; PTR*32 -> directory entry in DMA.
;           = FF        ; no match found.

getlst: push    d       ; save flag address
        xchg            ; pointer into DE
        mvi     c,17
        call    bdos
        pop     h
        mov     m,a     ; return flag
        inx     h
        xra     a
        mov     m,a
        ret

;*****************************************************************
; BDOS call 18 - search for next matching entry.  Call and
; return as for <getlst>.

getnxt: push    d       ; save flag address
        xchg            ; pointer into DE
        mvi     c,18
        call    bdos
        pop     h
        mov     m,a     ; return flag
        inx     h
        xra     a
        mov     m,a
        ret

;*****************************************************************
; BDOS call 26 - set DMA address.  Call with the syntax:
;       CALL SETDMA% (DMABUFF%(0))
; where DMABUFF is the first entry in a 128 byte array, ie set
;   with a DIM DMABUFF%(64) statement.

setdma: xchg            ; pointer into DE
        mvi     c,26
        call    bdos
        ret

        end
```

FIGURE 5

```
100 '          Directory reading program in Basic.
110 '
120 '    This program uses external assembly language routines
130 ' to assemble a string array containing matching entries
140 ' in the disc directory.  Note that no new variables may
150 ' be defined between calling GET1ST and the last call to
160 ' GETNXT.
170 '                                        Ron Kreymborg
180 DEFINT A-Z
190 DIM FCB(20),DMA(64),DIR$(100),CR(2)
200 DEF FNA(A!) = PEEK(A!)+PEEK(A!+1)*256-2^16
210 A! = PEEK(7)*256+3
220 N = PEEK(A!)
230 IF N <> 4 THEN STOP
240 SPLIT% = FNA(A!+1)
250 GET1ST% = FNA(A!+3)
260 GETNXT% = FNA(A!+5)
270 SETDMA% = FNA(A!+7)
280 FLG=0 : CR(0)=0 : CR(1)=0 : E=0 : K=0 : U=0 : L=0
290 '
300 PRINT "Enter required match string as 11 characters, with"
310 INPUT " a '?' to match any character: ",A$
320 PRINT
330 IF LEN(A$) <> 11 THEN PRINT "Pardon?" : GOTO 300
340 PRINT "Matching entries:"
350 FCB$ = "."+A$
360 FOR I=1 TO 100
370   DIR$(I) = "             "
380 NEXT I
390 '
400 ' Copy the specified filename into the FCB.
410 J = 0
420 FCB$ = FCB$+"     "
430 FOR I=0 TO 5
440   J = J+1 : M = 1 : GOSUB 660
450   J = J+1 : M = 256 : GOSUB 660
460 NEXT I
470 CALL SETDMA% (DMA(0))
480 CALL GET1ST% (FCB(0),FLG)
490 '
500 E = E+1
510 IF FLG>3 THEN 560
520 GOSUB 720
530 CALL GETNXT% (FCB(0), FLG)
540 GOTO 500
550 '
560 FOR K=1 TO E-1
570   IF (K-1) MOD 4 = 0 THEN PRINT
580   PRINT DIR$(K);"       ";
590 NEXT K
600 PRINT
610 PRINT
620 END
630 '
640 ' Load the ASCII sequence number into the correct
650 ' byte (specified by M) of the Ith FCB word.
660 B$ = MID$(FCB$,J,1)
670 IF B$ <> "." THEN FCB(I) = FCB(I)+ASC(B$)*M
680 RETURN
690 '
700 ' Extract each byte from the DMA buffer and load
710 ' them into the DIR$ string.
720 K = FLG*16
730 U = U+1
740 L = 1
750 FOR I=K TO K+5
760   CALL SPLIT% (DMA(I), CR(1), CR(0))
770   FOR J=0 TO 1
780     IF (I = K AND J = 0) THEN 810
790     MID$(DIR$(U),L,1) = CHR$(CR(J))
800     L = L+1
810   NEXT J
820 NEXT I
830 RETURN
```

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**

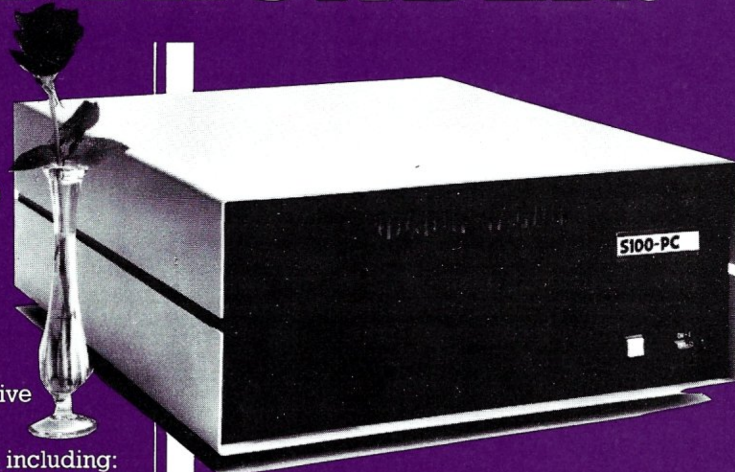
■ **RAM67 HIGH PERFORMANCE STATIC RAM**
High speed (100ns) low power CMOS static RAM. 128K bytes, extended addressing . . . .**PRICE $995.00**

■ **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 $1395.00**

■ **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

RP/M T.M.

By the author of Hayden's "CP/M Revealed."

New resident console processor RCP and new resident disk operating system RDOS replace CCP and BDOS without TPA size change.

User 0 files common to all users; user number visible in system prompt; file first extent size and user assignment displayed by DIR; cross-drive command file search; paged TYPE display with selectable page size. SUBMIT runs on any drive with multiple command files conditionally invoked by CALL. Automatic disk flaw processing isolates unuseable sectors. For high capacity disk systems RDOS can provide instantaneous directory access and delete redundant nondismountable disk logins. RPMPIP utility copies files, optionally prompts for confirmation during copy-all, compares files, archives large files to multiple floppy disks. RPMGEN and GETRPM self-install RP/M on any computer currently running CP/M®2.2. Source program assembly listings of RCP and RDOS appear in the RP/M user's manual.

RP/M manual with RPMGEN.COM and GETRPM.COM plus our RPMPIP.COM and other RP/M utilities on 8" SSSD $75. Shipping $5 ($10 nonUS). MC, VISA.

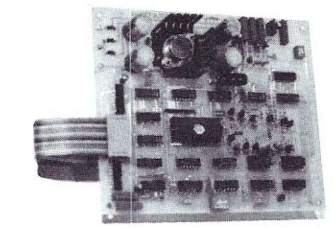
118 SW First St. - Box G
Warrenton, OR. 97146

Micro Methods, Inc.

(503) 861-1765

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/M.C. ACCEPTED –– $3.00 FOR COD

SMARTS RAM DISK SOFTWARE

for the QT, Great Salt Lake
California Digital or Computime
S-100 256K DRAM Memory Boards
Only
$40.00
or $500 with A&T board

UFDC-I Floppy Disk Controller
Reads/Writes 54 Formats
$275.00

CP/M Football Prediction Program
available in many formats
$40.00

All S-100 HARDWARE AVAILABLE IN BARE BOARDS KITS OR A&T

GSR COMPUTERS

60-10 69th St.
Maspeth, NY 11378
(718) 476-2091

NYS Residents Add Tax - $5 S&H

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.

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

Logical Name Translation: New Tricks for CP/M 2.2

by David Brewer

One of the most powerful features found in larger computer operating systems has somehow escaped the attention of the CP/M community entirely. Adding LOGICAL NAME TRANSLATION to enhance the CP/M man/machine interface is not difficult to implement or to understand: it is simply long overdue. The purpose of logical name translation is to simplify the dialogue between you and your operating system by "teaching" the system to recognize your own brand of shorthand. Having CP/M "Know what you mean" can free you from the standard syntax and command usage rules you have been putting up with all this time. After all, who's in charge here, you or your microcomputer? By making this feature resident in the operating system, translating "what you say" into "what you mean" takes only a few milliseconds and in no way affects any applications software that you run on your machine. This article presents a technique for implementing the translation feature on Z80-based CP/M 2.2 systems. The source listings are "low key" Z80 code (no use of index registers), so that adaptation for 8080 systems is not too difficult.

What is a logical name?

Logical names are string type variables that can be defined and used at the operating system command line level. Wouldn't it be nice to define DB to represent the string "DIR B:", and then be able to simply enter DB to produce a directory of disk B:? Logical names like DB are nothing more than a user-defined "shorthand" substitution for their string definitions. The beauty of the enhancement is that no disk access or any perceptible delay is introduced, since the translation tales are maintained in memory. No renaming of files or commands is involved, and standard command usage is still valid. Logical names can be defined to represent *any* string, whether it be an entire command such as DB, a portion of a command, a filename, or any series of printable characters. In essence, by making logical name definitions, you are teaching CP/M to recognize a wider variety of command syntax.

The logical names themselves are your own fabrication. On my system, "!" means "*.*", DEL and ERA mean the same thing, TY works the same as TYPE, and the list goes on. In 2K of memory, 50 to 100 definitions can be maintained, depending on string lengths. With a little imagination, you can see how CP/M could recognize commands in French, or execute lengthy PIP commands via a single keystroke. Handicapped users and Europeans take note!

How does it work?

The CCP portion of CP/M is responsible for analyzing command lines, processing built-in commands, loading user programs, and handling SUBMIT procedures. Collecting a line of input, however, is done by calling a subroutine in the BDOS. The translator intercepts this call so that the collected input line can be examined before the line is passed back to the CCP. If any logical names are detected on the input line, the translator replaces them with their string equivalents before returning control to the CCP. The translation is totally transparent to the CCP, which thinks you have entered a standard command line.

The catch to the enhancement is that the logical name translator must reside somewhere in memory at all times and not get trampled by applications programs. The only protected RAM space is that *above* the BIOS.

Making room in the attic

The logical name translator is going to cost 2K of memory, but it doesn't require any permanent changes to the operating system code itself, since it is installed on the fly. If you don't want to use the feature, you don't have to bring it into the environment.

Finding a place to hide the translator is the only problem to solve. Many systems already have RAM collecting dust in the "attic" above the BIOS due to disk controller or monitor ROMs in the 56K to 62K range. Phantom options on memory boards, or even an old fashioned 4K RAM board could be empoloyed to provide the 2K space. In the worst case, the MOVCPM and SYSGEN programs must be used to generate a new CP/M system that is 2K smaller in size. Before you wince at the thought, consider how many programs you have that won't run in 2K less space, and how often you will probably use the translation feature.

How do you use it?

Once the translator is loaded, logical names may be used freely on any CP/M command line. Communicating with the tanslator to define new logical names, delete old ones, or produce a list of currently active names is quite simple. If the first character of a command line, immediately following the CP/M prompt, is the sentinel character "*", then the remainder of the line will be treated as an INSTRUCTION to the translator. After the instruction is processed, an empty line will be forwarded to the CCP, which wouldn't know what to do with the instruction anyway. The translator does no disk access.

The instruction to define (or redefine) a logical name SC might look like this:

```
A>#SC:='STAT *.COM'  <ret>
```

In this case, SC has been chosen to represent an entire command to produce a status display of all .COM files on the disk. Notice that the logical name is followed by a colona nd equals sign, followed by its string definition enclosed in single quotes. From this point forward, any appearance of the name SC will cause the definition string to be substituted. The simple command:

```
A>SC  <ret>
```

will produce the dame status display as the standard command. If no blanks or special characters are included int he string, the single quote delimiters ar not required, as in:

```
A>#DEL:=ERA   <ret>
    or
A>#KILL:=DEL   <ret>
```

The second example shows that logical names can be defined in terms of other logical names. Now, ERA, DEL, and KILL can all be used interchangably to do the same thing. If you like a little class, try these:

```
A>#SHOW:=STAT   <ret>
A>#ALL:='*.*'   <ret>
A>#COMS:='*.COM'   <ret>
```

Now the status utility can be invoked with some finesse:

```
A>SHOW ALL   <ret>
    and
A>SHOW COMS   <ret>
```

My fingers never manage hit the W key on this one, so I gave in:

```
A>#SHOE:=SHOW   <ret>
```

and SHOE, SHOW, and STAT all look identical to my system.

If you are hacking away at a Fortran program called B:HALSTEAD.FOR, you might set up some definitions like these:

```
A>#F:='F80 B:HALSTEAD,=B:HALSTEAD.FOR'
A>#L:='L80 B:HALSTEAD,B:HALSTEAD/N/E'
```

Now, compiling is a one keystroke affair:

```
A>F   <ret>
```

and so in linking:

```
A>L   <ret>
```

You could, of course, put these commands in a submit procedure, and just to make it fun:

```
A>#GO:='SUBMIT B:HALSJOB'   <ret>
```

then crank up the procedure by simply typing:

```
A>GO   <ret>
```

Since extra spaces are generally acceptable on command lines, you can be creative with the Fortran compiler command:

```
A>#Z:='B:HALSTEAD'   <ret>
A>#COMPILE:='F80 ='   <ret>
```

and actually make the command readable (imagine that!)

```
A>COMPILE Z   <ret>
```

Some special effects

An interesting side effect is that semicolons and single-quote characters (normally invalid on comand lines) are always filtered out by the translator, but can be used as separators. Single quotes can be used to designate portions of a command to be excluded from translation, and semicolons can be used for concatenation of logical names to adjacent characters or other logical names. Consider:

```
A>#T:='TYPE'   <ret>
A>#XX:='B:ANIMAL'   <ret>
```

Then listing the Basic source file B:ANIMAL.BAS on the console becomes:

```
A>T XX;.BAS   <ret>
```

The exact substitution done by the translator would be:

```
A>TYPE B:ANIMAL;.BAS
```

but the semicolon is eliminated before the line is passed on to the CCP.

Logical names must always be separated from surrounding text by spaces, or the usual punctuation. The translator will never see a logical name imbedded in a filename, for instance, unless the semicolons or single quotes were used. The translator is smart enough to know the difference between a logical name like B and the drive specification B:, and doesn't attempt to translate any single characters that immediately precede a colon.

It is easy to get carried away with the translator, and you can get yourself into trouble if you redefine symbols like "=" or ":" or "*" to be logical names. If you make use of the USER feature of CP/M (you really should), then you might be tempted to do this:

```
A>#2:='USER 2'   <ret>
```

which will do just what you expect, but consider what will happen the next time you try:

```
A>SAVE 2 TEST.COM   <ret>
```

The exact translation would be:

```
A>SAVE USER 2 TEST.COM
```

and will be rejected by the CCP. The following definition is nearly as short, and works a little better.

```
A>#U2:='USER 2'   <ret>
```

Avoid defining a logical name in terms of itself, or your system will make a quick trip to Pluto.

The instruction to eliminate alogical name looks like:

```
A>#DROP KILL   <ret>
```

At this point, KILL is no longer defined.

The instruction to list all currently defined logical names and their string equivalences looks like:

```
A>#LOG   <ret>
```

which will display all the active names on the console.

Loading and saving logical names

It would not be very useful to require the user to retype all his definitions each time he powers up, so a means to batch load an entire set of disk-resident definitions is provided. This function is handled by the same program that activates the translator itself (Listing 2). Assuming that the program has been named LNT.COM, the translator is laoded by:

```
A>LNT   <ret>
```

and will remain active until a powerdown or reset occurs. An "OK" message is printed, indicting that the translator has been loaded. In this case, the current table of logical names will be initialized to empty. An o ptional filename may appear on the command line as in:

```
A>LNT MYNAMES.LOG   <ret>
```

In this case, the program will read the specified file, expecting to find ASCII text lines of logical name definitions exactly as you would enter them from your keyboard (including the leading "#" character). The table of logical name definitions will then be initialized to include the definitions from the file. If no filetype is given with the filename, and extention type of .LOG will be assumed. This command can be reissued at any time to load a new set of logical names or zero out the current table.

A second program (Listing 3) is used to save the currently active logical name definitions to a disk file. Assuming that the program has been named SAVLOG.COM, this function is accomplished by:

```
A>SAVLOG FILENAME.TYP   <ret>
```

The filename parameter is required and designates the file to receive the definitions. This file will be ASCII text compatible with the LNT.COM program, and may be edited, etc., using standard utilities. An extension type of .LOG is recommended.

The program in Listing 1 will help identify the exact memory locations that the translator program needs to assemble correctly. Run this program first to discover the locations of CTARG, BTARG1 to BTARG4, and WTARG1 to WTARG4 required in equate statements at the head of the main program in Listing 2.

If you are the least bit curious about this enhancement, I encourage you to try it out; it can save you a great deal of time and frustration, and it's fun!

FOR THE NON-DO-IT-YOURSELF-ER

The logical name translator is actually one part of a CP/M 2.2 enhancement package offered by a small user's group in Dallas, Texas. Another part of the package (in the same 2K space) is a command line editor that permits redisplay and modification of a previous command line in word-processor fashion. After corrections are made the line is resent for you. It is much more fun to fix mistkes than having to retype lines. An 8" SSSD disk with both features, a quick installation program, and manual is available. Send $18 and a description of your system and terminal c/o the author, P.O. Box 902306, Dallas, TX 75390-2306. All correspondance is welcomed.

Dave Brewer is a software test engineer at E-SYSTES in Garland, Texas, and is finishing an MS in computer science at the University of Texas at Dallas. He has been an assembler language devotee for eight years on a variety of machines, has taught mathematics and computer science, and is writing a book on ergonomic enhancement techniques for microcomputer systems.

LISTING 1. FINDEM Program

```
; This is the FINDEM program that will identify some specific memory
; locations needed for listing 2. This code will assemble directly
; using Microsoft's M80. Other assemblers may use DEFB and DEFW storage
; declarations instead of DB and DW.
;
; If it was necessary to use MOVCPM and SYSGEN to open up 2K space above
; the BIOS for the LOGICAL NAME TRANSLATOR, then this program should be
; run with the smaller system size.
;
; This program will scan the BIOS looking for references to the CCP and
; notify you what values should be used for some EQU statements at the
; start of the LNT program in listing 2.
;
        .Z80
        ASEG
        ORG 100H                        ; good place to start
CR      EQU       0DH                   ; carriage return
LF      EQU       0AH                   ; linefeed
;
START:  LD        DE,CTMSG              ; point to "CTARG IS " msg
        CALL      PRLINE                ; and send to console
        LD        HL,(0001)             ; get location of BIOS+3
        LD        DE,1603H-01A2H        ; calculate offset to expected CTARG
        XOR       A                     ; insure carry not set
        SBC       HL,DE                 ; put expected CTARG address in HL
        LD        A,5                   ; should be a 05 there
        CP        (HL)                  ; is it?
        JR        Z,CCPOK               ; if so, all is well, continue
        LD        DE,NFMSG              ; else point to "NOT FOUND" msg
        JP        PRLINE                ; send it and exit
;=================================================================
;NOTE:  If the NOT FOUND msg appears, then the CCP has been modified
;       or is not where it should be at 1600h bytes below the BIOS.
;       We are looking for a subroutine call ( CD 05 00 ) at location
;       CCP+01A1 in order to pick up the destination at CCP+01A2 and
;       CCP+01A3 which is needed by the LNT.
;=================================================================
;
CCPOK:  LD        A,H                   ; put the high byte of CTARG in A
        CALL      SENDAX                ; and send it to the console
        LD        A,L                   ; put the low byte of CTARG in A
        CALL      SENDAX                ; and send it to the console.
        LD        BC,0800H              ; search counter set to 2K bytes.
        LD        HL,(0001)             ; get location of BIOS+3
        LD        DE,1603H              ; get offset to CCP
        XOR       A                     ; insure carry not set
        PUSH      HL                    ; save the BIOS+3 location
        SBC       HL,DE                 ; HL will contain address of CCP
        EX        DE,HL                 ; so will DE.
        POP       HL                    ; retrieve location of BIOS+3
;=================================================================
; Start of BIOS scan loops. We are looking for BIOS references to
; the CCP in LOOP1, and references to CCP+3 in LOOP2.
;=================================================================
;
LOOP1:  LD        A,E                   ; put low byte of CCP address in A
        CP        (HL)                  ; match in the BIOS?
        JR        NZ,GOON               ; if not, go on
        INC       HL                    ; else check the next byte
        LD        A,D                   ; to see if the high byte also matches
        CP        (HL)                  ; does it?
        JR        NZ,GOON               ; if not, go on
        DEC       HL                    ; else we found a BTARG...point the low byte
        DEC       HL                    ; Point to previous byte.
        CALL      ISJMP                 ; Check to see if this is a jump.
        INC       HL                    ; Look at the low byte again.
        JR        NZ,GOON               ; If not type of jump, forget this one.
        PUSH      BC                    ; save the counter
        PUSH      DE                    ; and the CCP address
        LD        DE,GOTBT              ; get ready to identify the BTARG.
        CALL      PRLINE                ; send the "BTARG" msg,
```

```
        LD      A,(COUNT)        ; put 1,2,3 or 4 in A
        INC     A                ; increment the BTARG counter,
        LD      (COUNT),A        ; and store it away.
        CALL    SENDA            ; and send the counter to console
        LD      DE,AT            ; point to " AT " string
        CALL    PRLINE           ; and send that,
        LD      A,H              ; point to high byte of this BTARG
        CALL    SENDAX           ; and tell us what it is,
        LD      A,L              ; point to low byte,
        CALL    SENDAX           ; and tell us what it is.
        POP     DE               ; retrieve CCP address.
        POP     BC               ; and the scan counter,
GOON:   INC     HL               ; move on along to next BIOS byte,
        DEC     BC               ; have we scanned 2K yet?
        LD      A,C              ; if not,
        OR      B                ; then
        JR      NZ,LOOP1         ; continue scanning for CCP matches.
        LD      BC,0800          ; done with first scan. Reset counter.
        LD      A,'0'            ; and the target id counter.
        LD      (COUNT),A        ; store it.
        LD      HL,(0001)        ; now point to BIOS+3 once again
        INC     DE               ; this time we look for
        INC     DE               ; reference to CCP+3, so
        INC     DE               ; DE in incremented by 3
LOOP2:  LD      A,E              ; put low byte of CCP+3 in A
        CP      (HL)             ; match in BIOS?
        JR      NZ,CONT          ; if not, continue
        INC     HL               ; looks good so far,
        LD      A,D              ; check the high byte.
        CP      (HL)             ; found CCP+3 reference?
        JR      NZ,CONT          ; if not, continue
        DEC     HL               ; found a WTARG..back up to 1st byte.
        DEC     HL               ; Point to previous byte.
        CALL    ISJMP            ; Check to see if this is a jump.
        INC     HL               ; Look at the low byte again.
        JR      NZ,CONT          ; If not type of jump, forget this one.
        PUSH    BC               ; save the scan counter
        PUSH    DE               ; and CCP+3 location
        LD      DE,GOTWT         ; point to " WTARG " string

        CALL    PRLINE           ; and send that,
        LD      A,(COUNT)        ; put 1,2,3 or 4 in A
        INC     A                ; bump it for next time
        LD      (COUNT),A        ; and store it away
        CALL    SENDA            ; send which WTARG number this is.
        LD      DE,AT            ; point to " AT " string
        CALL    PRLINE           ; and send the string
        LD      A,H              ; put high byte of this WTARG in A
        CALL    SENDAX           ; and send it to console
        LD      A,L              ; ditto for the low byte
        CALL    SENDAX           ; to tell us where this WTARG is.
        POP     DE               ; retrieve CCP+3 location
        POP     BC               ; and the scan counter.
CONT:   INC     HL               ; point to next BIOS byte.
        DEC     BC               ; have we checked 2K bytes yet?
        LD      A,C              ; if not
        OR      B                ; then
        JR      NZ,LOOP2         ; keep scanning.
        JP      Z,0000           ; if so, then all done.
;
;==================================================================
; NOTE: It may very well be the case that no BTARG msg is sent, or no
;       WTARG msg is sent, but at least ONE msg of either type should
;       appear. As many as 4 msgs of each type may be sent. If more than
;       four jumps to the CCP or CCP+3 occur in your BIOS, you better
;       write me a letter.
;==================================================================
;
; Subroutine to determine if the byte pointed to by HL is a type of JUMP.
;
ISJMP:  LD      A,(HL)           ; Load the byte at HL.
        CP      0C3H             ; If its an unconditional jump,
        RET     Z                ; then return.
        CP      0DAH             ; Else, is it jump on carry?
        RET     Z                ; If so, then return.
        CP      0D2H             ; Is it jump on no carry?
        RET     Z                ; If so return.
        CP      0CAH             ; Is it jump on zero?
        RET     Z                ; If so return.
        CP      0C2H             ; Is it jump on non-zero?
        RET                      ; If so return zero, else non-zero.
;
;
; Subroutine to send a string to the console.
;
PRLINE: PUSH    HL
        LD      C,9
        CALL    0005
        POP     HL
        RET
;
; Subroutine to print character in A on console.
;
SENDA:  PUSH    HL
        LD      E,A

        LD      C,2
        CALL    0005
        POP     HL
        RET
;
; Subroutine to send the two character ASCII hex of value in A
;
SENDAX: PUSH    AF               ; save A
        RRCA                     ; shift the high nibble
        RRCA                     ; down to
        RRCA                     ; the
        RRCA                     ; low nibble
        CALL    ZIP              ; go the low nibble
        POP     AF               ; retrieve the original value
;
; we will fall through to the ZIP routine for the second nibble
;
; This little piece of code takes a value from 0 to F and produces the
; correct ASCII code to print '0' to 'F'. I love it.
;
ZIP:    AND     0FH              ; mask off the high nibble
        ADD     A,90H            ; perform magic
        DAA                      ; once,
        ADC     A,40H            ; perform magic
        DAA                      ; twice,
        CALL    SENDA            ; and send it.
        RET
;
;
CTMSG:  DB      CR,LF,'CTARG IS $'
NFMSG:  DB      'NOT FOUND!$'
GOTBT:  DB      CR,LF,'BTARG$'
GOTWT:  DB      CR,LF,'WTARG$'
AT:     DB      ' AT $'
COUNT:  DB      '0'
        END
```

LISTING 2. LOGICAL NAME TRANSLATOR program

```
; This is the LOGICAL NAME TRANSLATOR program. Care has been taken to avoid
; any uncommon Z80 pseudo ops, except perhaps the one on the first line.
; The only Z80 specific instructions are the block moves and relative jumps
; so that adapting to 8080 machines requires minimal effort. This code will
; assemble directly using Microsoft's M80. Other assemblers may use DEFB
; and DEFW storage declarations instead of DB and DW.
;
        .Z80
        ASEG
        ORG 100H
;
```

```
;=====================================================================
; THE VALUE OF DTOUR MUST BE CHANGED TO BE THE START OF THE 2K FREE RAM
; ABOVE THE BIOS. IT IS CURRENTLY SET TO 58K.
;=====================================================================
;
DTOUR  EQU     ØE8ØØH          ; Points to 2K RAM space above BIOS
;
OFF    EQU     DTOUR - SCODL   ; offset value used everywhere
LASTAD EQU     DTOUR + 2Ø46    ; end of the logical name table
CPMIO  EQU     5
;
;=====================================================================
; SOME OF THE FOLLOWING EQU VALUES MUST BE CHANGED IF IDENTIFIED BY THE
; FINDEM PROGRAM IN LISTING 1. THOSE NOT SPECIFIED BY THAT PROGRAM SHOULD
; BE LEFT ALONE.
;=====================================================================
;
CTARG  EQU     ØBEA2H          ; this one will be changed for sure.
BTARG1 EQU     ØD3C8H          ; modify if identified by FINDEM program.
BTARG2 EQU     KEEP+OFF        ; modify if identified by FINDEM program.
BTARG3 EQU     KEEP+OFF        ; modify if identified by FINDEM program.
BTARG4 EQU     KEEP+OFF        ; modify if identified by FINDEM program.
WTARG1 EQU     ØD3CBH          ; modify if identified by FINDEM program.
WTARG2 EQU     KEEP+OFF        ; modify if identified by FINDEM program.
WTARG3 EQU     KEEP+OFF        ; modify if identified by FINDEM program.
WTARG4 EQU     KEEP+OFF        ; modify if identified by FINDEM program.
;
START: LD      HL,(BTARG1)     ; sorry, but a little self modifying code
       LD      (FIXX1+1),HL    ; to deal with unknowns is a must...
       LD      HL,(WTARG1)     ; but only
       LD      (FIXX2+1),HL    ; twice.
       LD      HL,(CTARG)      ; pick up the subroutine call in the CCP
       LD      DE,DOOR+OFF     ; point to our own destination instead,
       XOR     A               ; insure carry not set
       SBC     HL,DE           ; and see if the LNT is already active.
       JR      Z,READY         ; If so, then we can skip the next part,
       LD      HL,SCODL        ; else get ready to move the code
       LD      DE,DTOUR        ; to the 2K space above the BIOS.
       LD      BC,ECODL-SCODL  ; Get length of the code
       LDIR                    ; and do the block move.
       LD      HL,DTOUR        ; Point to the entry address
BIO1:  LD      (BTARG1),HL     ; and modify the BIOS to intercept warm
       LD      (BTARG2),HL     ; starts to the CCP in all the right
       LD      (BTARG3),HL     ; places found by FINDEM program.
       LD      (BTARG4),HL     ; Should never be this many of them.
       LD      HL,DTOUR+3      ; All BIOS jumps to CCP+3 will now go to
BIO2:  LD      (WTARG1),HL     ; DTOUR+3, too, in case your BIOS does
       LD      (WTARG2),HL     ; any fancy stuff. There really shouldn't
       LD      (WTARG3),HL     ; be more than one or two, but you never
       LD      (WTARG4),HL     ; know.
READY:
       LD      A,(8ØH)         ; Get length of command line extras,
       OR      A               ; if not zero
       JR      NZ,RDLOG        ; then go read the file in
RBYE:  LD      (LNT+OFF),A     ; else no file given, put a zero in LNT.
       JP      GCLIP           ; and go say goodbye.
RDLOG: LD      A,(ØØ65H)       ; See if there was a extention on the
       CP      ' '             ; file name. Is extention blank?
       JR      NZ,GOODN        ; If not, never mind, whole name given.
       LD      A,'L'           ; Else fill it in
       LD      (ØØ65H),A       ; with an L
       LD      A,'O'           ; and
       LD      (ØØ66H),A       ; with an O
       LD      A,'G'           ; and
       LD      (ØØ67H),A       ; with a G.
GOODN: LD      DE,ØØ5CH        ; Ready to read it, point to FCB,
       LD      C,ØFH           ; set up the OPEN function,
       CALL    CPMIO           ; and open the file.
       CP      ØFFH            ; a problem (like not there)?
       JR      NZ,DORD         ; If not, then ready to load it in,
       LD      DE,NOTF         ; else point to NOT FOUND,
       LD      C,9             ; set up to PRINT STRING,
       CALL    CPMIO           ; send to console,
       JR      RBYE            ; and abort the program.
DORD:  LD      DE,LDMA         ; We want to load the file at LDMA
RDLP:  PUSH    DE              ; save that address
       LD      C,1AH           ; set up the SET DMA function
       CALL    CPMIO           ; and tell CPM where it goes.
       LD      DE,ØØ5CH        ; Point to the FCB,
       LD      C,14H           ; set up the READ function,
       CALL    CPMIO           ; and pull in a sector from disk.
       OR      A               ; All done?
       JR      NZ,RDONE        ; Yes! then get out of the read loop
       POP     HL              ; else retrieve the DMA address
       LD      DE,ØØ8ØH        ; bump it by 128
       ADD     HL,DE           ; and now we have a new DMA address.
       EX      DE,HL           ; Give it to DE
       JR      RDLP            ; and go read another sector.
RDONE: POP     HL              ; Retrieve the last DMA address
       LD      A,1AH           ; and put a CPM end-of-file mark
       LD      (HL),A          ; there just to be safe.
       LD      HL,LDMA         ; Point to start of file in memory,
SYNTH: LD      DE,FAKE+2       ; point to our CCP look-alike buffer,
       LD      BC,ØØØDH        ; store a carriage return in C, Ø in B,
MLOOP: LD      A,(HL)          ; look at a byte from the file.
       CP      C               ; Is it carriage return?
       JR      Z,MDONE         ; If so, stop filling the buffer,
       INC     B               ; else count how many bytes we've moved
       LD      (DE),A          ; and put a byte from the file into buffer.
       INC     DE              ; Bump the buffer pointer,
       INC     HL              ; bump the file pointer,
       JR      MLOOP           ; and go look at the next byte.
MDONE: INC     HL              ; Should be a linefeed after the return
       INC     HL              ; and the next byte is start of new line.
       PUSH    HL              ; Save the file pointer,
       LD      HL,FAKE+1       ; point to byte count location in buffer,
       LD      (HL),B          ; and put the length of the line there.
       DEC     HL              ; Back up to very start of buffer
       LD      (BUFØ+OFF),HL   ; pretend we are calling from the CCP !
       CALL    BDOOR+OFF       ; and take a shortcut into the translator.
       POP     HL              ; LNT updated his table, now back to the file
       LD      A,(HL)          ; Get the first byte from the next line.
       CP      1AH             ; All done by any chance?
       JR      NZ,SYNTH        ; No, go process another line.
       JP      GCLIP           ; All done. Go say goodbye.
FAKE:  DB 127                  ; Start of CCP buffer look-alike,
       DS 127                  ; with same length.
GCLIP: LD      DE,OKMSG        ; Point to OK message.
       LD      C,9             ; Send the message at DE,
       CALL    CPMIO           ; and AU REVOIR !
       JP      ØØØØ
OKMSG: DB      '  OK$'
;
; main segment to be relocated to high memory begins here..
;
SCODL: JP      $MAIN+OFF                ; Entry from BIOS (JMP CCP)
PLUS3: JP      $MAIN2+OFF               ; Entry from BIOS (JMP CCP+3)
BUFØ:  DW      Ø                        ; Storage for CCP's buffer address
KEEP:  DW      Ø                        ; Storage for general use
FIN:   DW      Ø                        ; Storage for end of buffer address
NOTF:  DB 13,1Ø,'NOT FOUND',13,1Ø,'$'
NOWAY: DB 13,1Ø,'NO ROOM',13,1Ø,'$'
IERR:  DB 13,1Ø,'NO TRANSLATION..RE-ENTER.',13,1Ø,'$'
$LOG:  DB 4,'#LOG'
$DROP: DB 5,'#DROP'
       DS      24                       ; Local stack space
SAVE:  DW      Ø                        ; Storage for CP/M's stack pointer
$MAIN: LD      HL,DOOR+OFF              ; Initialize the CCP on warm boot
       LD      (CTARG),HL               ; by altering the BDOS call
FIXX1:  JP      ØØØØ                    ; This will be a JMP to CCP
;
$MAIN2: LD     HL,DOOR+OFF              ; Initialize CCP on fancy boot
       LD      (CTARG),HL               ; by altering the BDOS call
FIXX2: JP      ØØØØH                    ; This will be a JMP to CCP+3
;
; THE ADDRESS OF THE TABLE IS PUT AT DOOR-2 SO THAT SAVLOG CAN FIND IT BY
; CHECKING THE CALL ADDRESS FROM THE CCP AND SUBTRACTING 2.
;
```

```
        DW      LNT+OFF                 ; Keep start of table for SAVLOG
;
; HERE IS WHERE THE CCP WILL CALL WHEN HE WANTS AN INPUT LINE
;
DOOR:   LD      (BUFØ+OFF),DE           ; Save CCP's buffer address
        CALL    CPMIO                   ; then go get the input line.
BDOOR:  LD      HL,(BUFØ+OFF)           ; HL gets the start of buffer
        INC     HL                      ; plus one (length of line).
        LD      A,(HL)                  ; Did you just type <ret>?
        OR      A                       ; If so, then theres nothing to
        RET     Z                       ; to look at..return to CCP.
        LD      D,H                     ; else let DE point to line length,
        LD      E,L                     ; then add the length to DE so
        CALL    ADE+OFF                 ; that it points to end of line,
        INC     DE                      ; plus one, then
        XOR     A                       ; put a ØØ byte there to
        LD      (DE),A                  ; terminate the input line, and
        LD      (FIN+OFF),DE            ; store the end-of-line address.
        PUSH    HL                      ; Save HL (buffer start+1).
LCLP:   INC     HL                      ; Point to next byte of line,
        LD      A,(HL)                  ; and check to see
        OR      A                       ; if its the ØØ terminator,
        JR      Z,CHKOV                 ; If so, the check is over, jump.
        CP      'a'                     ; Is the character lower case?
        JR      C,LCLP                  ; If not, continue
        CP      '{'                     ; else this code will convert to
        JR      NC,LCLP                 ; uppercase. (The CCP does the
        AND     ØDFH                    ; same thing later, but we need
        LD      (HL),A                  ; to do it NOW).
        JR      LCLP                    ; Continue converting to upper.
CHKOV:  POP     HL                      ; Retrieve start of bufer+1.
        INC     HL                      ; point to first byte of line.
        LD      A,(HL)                  ; Is this an INSTRUCTION line?
        CP      '#'                     ; If so, hey..wait we better
        LD      (SAVE+OFF),SP           ; set up our own stack in either
        LD      SP,SAVE+OFF             ; case. Save his Stack pointer.
        JP      Z,ANLIZ+OFF             ;...If so, then go process line.
;
; THIS IS NOT A LNT INSTRUCTION
;
        EX      DE,HL                   ; DE gets start of buffer
SCNLP:  CALL    GETWD+OFF               ; get the piece of input at DE
        JR      Z,DONE                  ; If no more pieces, get out.
        LD      A,(DE)                  ; else look at last char scanned.
        CP      ':'                     ; Was the seperation char a : ?
        JR      NZ,NOTD                 ; If not, never mind, jump NOTD
        LD      A,1                     ; else see if piece was only
        CP      B                       ; one character (maybe drive spec).
        JR      Z,SCNLP                 ; if so, ignore this piece, loop.
NOTD:   PUSH    DE                      ; Got something. Save line ptr.
        CALL    FIND+OFF                ; go see if its in table.
        CALL    NZ,XLATE+OFF            ; If it is, do translation.
        POP     DE                      ; If not, remember where we were,
        JR      SCNLP                   ; and continue scannin the line.
DONE:   LD      HL,(BUFØ+OFF)           ; Done scanning, point HL to buffer
        INC     HL                      ; plus one, and
        INC     HL                      ; finally to line length byte.
        LD      BC,273BH                ; Load a ; and ' into B and C.
CLEAN:  LD      A,(HL)                  ; Put byte from line into A.
        OR      A                       ; Are we at end-of-line?
        JP      Z,BYE+OFF               ; If so, then go wind down,
        CP      B                       ; else check for ;
        JR      Z,CHOP1                 ; If ; found, go eliminate it.
        CP      C                       ; else check for '
        JR      Z,CHOP1                 ; If ' found, go eliminate it.
        INC     HL                      ; point to next byte in line,
        JR      CLEAN                   ; and continue cleaning up.
CHOP1:  CALL    ERA1+OFF                ; A ; or ' found, delete it.
        JR      CLEAN                   ; and continue cleaning up.
;
; ERA1.  DELETE ONE CHARACTER POINTED TO BY HL, AND AJUST LINE LEN.
;
ERA1:   PUSH    HL                      ; Save
        PUSH    DE                      ; every-
        PUSH    BC                      ; body.
        LD      DE,(FIN+OFF)            ; DE looks at end-of-line.
        EX      DE,HL                   ; Now HL does, DE to target byte.
        XOR     A                       ; Insure carry not set,
        SBC     HL,DE                   ; and find out how far from end.
        LD      B,H                     ; Byte count to end goes into
        LD      C,L                     ; register BC.
        PUSH    DE                      ; Copy the address of byte to
        POP     HL                      ; delete into HL,
        INC     HL                      ; point to byte following,
        LDIR                            ; and shift the whole line down.
        LD      HL,(FIN+OFF)            ; Get the old end-of-line address,
        DEC     HL                      ; subtract one,
        LD      (FIN+OFF),HL            ; to update it.
        LD      HL,(BUFØ+OFF)           ; Look back at the start of buffer
        INC     HL                      ; plus one (line length),
        DEC     (HL)                    ; and update the length.
        POP     BC                      ; Bring
        POP     DE                      ; everybody
        POP     HL                      ; back, and
        RET                             ; return.
;
; XLATE. Replaces the logical name with its string equivalent.
;
XLATE:  LD      A,(DE)                  ; A gets length of logical name
        LD      B,A                     ; from table, then copy to B.
DELLP:  CALL    ERA1+OFF                ; Delete the logical name from the
        DJNZ    DELLP                   ; input line, char by char.
        LD      A,(DE)                  ; A gets length once again,
        CALL    ADE+OFF                 ; add to DE to point to the string
        INC     DE                      ; definition length in the table.
        PUSH    HL                      ; Save pointer into input line.
        PUSH    DE                      ; Save pointer to string in table.
        LD      A,(DE)                  ; A gets replacement string length,
        LD      B,A                     ; and so does B ( loop counter ).
ADDLP:  CALL    ADD1+OFF                ; Insert extra bytes into the input
        DJNZ    ADDLP                   ; line to make room for string.
        POP     HL                      ; Retrieve pointer to string length
        LD      C,(HL)                  ; from table and copy into C so
        LD      B,Ø                     ; that BC is byte counter of move.
        INC     HL                      ; Now point to the string itself,
        POP     DE                      ; get the pointer into input line.
        PUSH    DE                      ; Keep a copy on the stack.
        LDIR                            ; Block move the string into line.
        POP     DE                      ; Revive the pointer into the line,
        POP     BC                      ; This is the return address!!
        POP     HL                      ; We are replacing the input line
        PUSH    DE                      ; pointer deep on the stack.
        PUSH    BC                      ; Restore the return address,
        RET                             ; and the translation is done.
;
; ADD1  This routine duplicates the byte at HL on the input line and
;       slides the rest of the line down to make room.
;
ADD1:   PUSH    HL                      ; Save
        PUSH    DE                      ; every-
        PUSH    BC                      ; body.
        LD      DE,(FIN+OFF)            ; DE points to ØØ at end of line.
        EX      DE,HL                   ; Now HL does. DE points into line.
        XOR     A                       ; insure carry not set.
        SBC     HL,DE                   ; How many bytes must be shifted?
        LD      B,H                     ; BC will contain
        LD      C,L                     ; the byte count.
        INC     BC                      ; (Be sure to move the ØØ too).
        LD      HL,(FIN+OFF)            ; HL points to ØØ at end of line.
        PUSH    HL                      ; Save this spot while we
        INC     HL                      ; update this pointer
        LD      (FIN+OFF),HL            ; by one location.
        POP     DE                      ; Now end of line goes to DE,
        EX      DE,HL                   ; swapped with HL, and
        LDDR                            ; the whole line is shifted by one.
ADONE:  LD      A,(DE)                  ; If the last byte moved was not
        OR      A                       ; the ØØ (we were at end of line)
```

```
        JR      NZ,ABYE                 ; then jump to ABYE.
        LD      A,'&'                   ; Else we have 2 00 bytes.
        LD      (DE),A                  ; Generate a non-zero dummy byte.
ABYE:   LD      HL,(BUF0+OFF)           ; Point to input line length
        INC     HL                      ; and update it by adding one
        INC     (HL)                    ; for the byte we just added.
        POP     BC                      ; Restore
        POP     DE                      ; every-
        POP     HL                      ; body.
        RET                             ; and return.
;
; THIS WAS A LNT COMMAND...Sentinel character detected in column 1.
;
ANLIZ:  EX      DE,HL                   ; Point DE to start of input line.
        CALL    GETWD+OFF               ; Pick up the first word of input,
        PUSH    DE                      ; and save address of terninator.
        LD      DE,$LOG+OFF             ; Point to "#LOG" string.
        CALL    TEST+OFF                ; Does that match the input?
        JR      NZ,DCHCK                ; If not go check for DROP.
;
; COMMAND WAS #LOG
;
        CALL    CRLF+OFF                ; Send CR-LF to console.
        LD      DE,LNT+OFF              ; point to start of table.

LOGLP:  LD      A,(DE)                  ; get next entry offset byte.
        OR      A                       ; Is it zero (end of table)?
        JP      Z,IGNOR+OFF             ; If so we are done. Jump out.
        INC     DE                      ; Else point to length of name,
        CALL    SSTR+OFF                ; and go send the name to console.
        LD      A,':'                   ; Now send a colon to
        CALL    SEND1+OFF               ; follow the name, and
        LD      A,'='                   ; then an = to make it
        CALL    SEND1+OFF               ; look nice on the screen.
        INC     DE                      ; Point to length of definition
        CALL    SSTR+OFF                ; string and go send the string.
        CALL    CRLF+OFF                ; Send a CR-LF
        INC     DE                      ; and we should be at start of next
        JR      LOGLP                   ; table entry. Loop until at end.
;
DCHCK:  LD      DE,$DROP+OFF            ; Point to "#DROP" string.
        CALL    TEST+OFF                ; Is that the instruction?
        JP      NZ,NEWID+OFF            ; If not, go test for new name.
;
;COMMAND WAS #DROP
;
        POP     DE                      ; Retrieve pointer to next word on
        CALL    GETWD+OFF               ; input line and go get it.
        JR      Z,NFMSG                 ; If nothing there, goto error.
        CALL    FIND+OFF                ; Else hunt the name in the table.
        JR      Z,NFMSG                 ; If not in table, goto error.
        CALL    KILL1+OFF               ; Otherwise eliminate table entry.
        JP      IGNOR+OFF               ; We are done. Get out.
IMSG:   LD      DE,IERR+OFF             ; Point to "COMMAND ERROR"
        JR      SPILL                   ; Go print it.
NFMSG:  LD      DE,NOTF+OFF             ; Point to "NOT FOUND"
SPILL:  LD      C,9                     ; Print the message
        CALL    CPMIO                   ; and
        JP      IGNOR+OFF               ; get out.
;
; TEST FOR NEW IDENTIFIER.
;
NEWID:  POP     DE                      ; Retrieve pointer to word term-
        LD      A,(DE)                  ; inator from input line.
        CP      ':'                     ; Was it a colon?
        JR      NZ,IMSG                 ; If not, ?????, go tell him ??
; WAS NEW IDENTIFIER..
        PUSH    DE                      ; Save pointer to :
        INC     HL                      ; Point to byte after # char.
        PUSH    HL                      ; and save that location.
        DEC     B                       ; Adjust length of new name,
        PUSH    BC                      ; and save it.
        CALL    FIND+OFF                ; Hunt the name in the table.
        CALL    NZ,KILL1+OFF            ; and eliminate it if found.
BUILD:  LD      DE,LNT+OFF              ; Point to start of table,
        POP     BC                      ; and retrieve the name length.
ENDLP:  LD      A,(DE)                  ; Look at this byte from table.
        OR      A                       ; Is it 0 (at end of table)?
        JR      Z,GEND                  ; If so, stop looking for end.
        INC     DE                      ; Else keep looking
        JR      ENDLP                   ; for end of table.
GEND:   LD      (FIN+OFF),DE            ; FIN can be used for temp storage.
        INC     DE                      ; Point to byte after end of table.
        CALL    FULLCK+OFF              ; See if we have room for this entry.
        LD      A,B                     ; A gets length of new name.
        LD      (DE),A                  ; Store into table,
        INC     DE                      ; point to next table byte.
        POP     HL                      ; Retrieve pointer to name in line.
        CALL    CPYLP+OFF               ; and put a copy into the table.
        LD      (KEEP+OFF),DE           ; Store where we are in the table.
        POP     DE                      ; Retrieve pointer to : in line.
        CALL    GETWD+OFF               ; Pick up the definition string.
        JR      Z,IMSG                  ; If not any, send error and out.
        LD      DE,(KEEP+OFF)           ; Where were we in the table..
        CALL    FULLCK+OFF              ; See if we have room in the table.
        LD      A,B                     ; A gets length of string def.
        LD      (DE),A                  ; Store into the table,
        INC     DE                      ; and follow the length with a
        CALL    CPYLP+OFF               ; copy of the string definition.
        XOR     A                       ; A is 0.
        LD      (DE),A                  ; Make a new end of table marker.
        LD      HL,(FIN+OFF)            ; Retrieve old marker (still 0),
        EX      DE,HL                   ; Swap these addresses.
        SBC     HL,DE                   ; How many bytes did we add?
        LD      A,L                     ; Put answer in A (less than 255).
        LD      (DE),A                  ; and udate the old marker.
        JP      IGNOR+OFF               ; We are done. Get out.
;
; GENERAL PURPOSE SENDSTRING ROUTINE. DE POINTS TO LEN+STRING BUFFER
;
SSTR:   LD      A,(DE)                  ; A gets length of string to send.
        PUSH    BC                      ; Save whatever is in BC.
        LD      B,A                     ; B will be byte counter.
SSLP:   INC     DE                      ; Point to next byte to print.
        LD      A,(DE)                  ; Put it into A,
        CALL    SEND1+OFF               ; and send it to console.
        DJNZ    SSLP                    ; Loop until done.
        POP     BC                      ; Restore BC.
        RET                             ; Bye.
;
; CPYLP ..BLOCK MOVE STRING AT HL TO LOCATION DE. B HAS THE LENGTH.
;
CPYLP:  LD      A,(HL)                  ; Get a byte.
        LD      (DE),A                  ; Copy it.
        INC     HL                      ; Increment source and
        INC     DE                      ; destination addresses
        DJNZ    CPYLP                   ; and continue.
        RET
;
; KILL1. DROP AN ENTRY. DE POINTS TO IDENTIFIER LEN IN THE TABLE.
;
KILL1:  DEC     DE                      ; Back up to start of entry,
        PUSH    DE                      ; and copy this address
        POP     HL                      ; into HL.
        LD      A,(DE)                  ; A is length of this entry.

        CALL    ADE+OFF                 ; Find the start of next entry,
        PUSH    HL                      ; Save the start of entry to delete.
        LD      HL,LASTAD               ; Point to end of table space.
        SBC     HL,DE                   ; Compute number of bytes to move.
        PUSH    HL                      ; and copy into
        POP     BC                      ; BC.
        POP     HL                      ; Retrieve start of entry to delete.
        EX      DE,HL                   ; Swap with next entry address.
        LDIR                            ; and shift the entire table down.
        RET
```

```
; SEND CHARACTER IN A TO CONSOLE
;
SEND1:  PUSH    DE                      ; Save
        PUSH    HL                      ; every-
        PUSH    BC                      ; body.
        LD      C,2                     ; Function number for conout.
        LD      E,A                     ; CP/M likes it in register E.
        CALL    CPMIO                   ; Send it.
        POP     BC                      ; Restore
        POP     HL                      ; every-
        POP     DE                      ; body, and
        RET                             ; return.
;
; SEND CR-LF TO CONSOLE
;
CRLF:   LD      A,13                    ; 13 is a carriage return.
        CALL    SEND1+OFF               ; Send it.
        LD      A,10                    ; 10 is a line feed.
        CALL    SEND1+OFF               ; Send it.
        RET
;
; GETWD ROUTINE. DE POINTS TO STRING AT ENTRY, AT TERMINATOR ON EXIT.
;               HL POINTS TO START OF WORD, B IS LENGTH, C IS 1 IF
;               A QUOTED STRING, ELSE 0. ZERO SET IF END OF BUFFER.
;
GETWD:
        LD      H,D                     ; Put a copy of DE
        LD      L,E                     ; into HL, and
        LD      BC,0000                 ; zeroes in B and C.
        DEC     HL                      ; Syncronize the loop.
LEAD:   INC     HL                      ; Point to next byte of line,
        LD      A,(HL)                  ; and see if at
        OR      A                       ; end of line (00 byte).
        RET     Z                       ; If so, goodbye.
        CALL    PUNCK+OFF               ; Else look for punctuation, and
        JR      Z,LEAD                  ; skip over any you find.
        CP      39                      ; Check for single quote.
        JR      NZ,GSTRT                ; If not quoted, never mind, jump.
        INC     C                       ; Else set the quoted string flag.
GSTRT:  LD      D,H                     ; DE points to first byte of word,
        LD      E,L                     ; just like HL.
GCONT:  INC     B                       ; bump the word length counter.
        INC     HL                      ; Point to next byte on the line.
        LD      A,(HL)                  ; and see if we are
        OR      A                       ; at end of line (00 byte).
        JR      Z,OUT                   ; If so, stop looking, jump.
        LD      A,C                     ; Else check quoted string flag.
        OR      A                       ; If quotes present, then
        JR      NZ,QCK                  ; don't bother checking, jump.
        LD      A,(HL)                  ; Otherwise, look for terminating
        CALL    PUNCK+OFF               ; punctuation and
        JR      Z,OUT                   ; jump if thats the case.
QCK:    CP      39                      ; Check for terminating quote.
        JR      NZ,GCONT                ; If not found, keep looking. Loop.
        INC     B                       ; Got terminating quote.
        INC     HL                      ; Point to byte following.
OUT:    EX      DE,HL                   ; HL gets start. DE gets end of word.
        INC     A                       ; Insure zero not set,
        RET                             ; and return.
;
; PUNCK. CHECKS FOR LEADING OR TRAILING PUNCTUATION.
;
PUNCK:  CP      ' '
        RET     Z
        CP      ','
        RET     Z
        CP      ':'
        RET     Z
        CP      '='
        RET     Z
        CP      ';'
        RET     Z
        CP      '+'
        RET     Z
        CP      '-'
        RET     Z
        CP      '['
        RET     Z
        CP      ']'
        RET     Z
        CP      '/'
        RET     Z
        RET
;
; FIND. SCANS LNT UNTIL A MATCH IS FOUND FOR STRING AT (HL). RETURNS
; ZERO IF NO MATCH, ELSE DE POINTS TO LEN BYTE OF ID STRING.
;
FIND:   LD      DE,LNT+OFF              ; Point to start of table.
FLOOP:  LD      A,(DE)                  ; Get first byte of entry.
        OR      A                       ; If at end of table,
        RET     Z                       ; return with zero set.
        INC     DE                      ; Else point to name length byte.
        CALL    TEST+OFF                ; Compare with string at HL.
        JR      Z,GOTIT                 ; If a match, jump to GOTIT.
FNXT:   DEC     DE                      ; Point to first byte of entry,
        LD      A,(DE)                  ; pick up the entry length,
        CALL    ADE+OFF                 ; and move the pointer to next one.
        JR      FLOOP                   ; Continue checking for matches.
GOTIT:  INC     A                       ; Match! Insure zero not set,and
        RET                             ; return.
;
; TEST. GENERAL PURPOSE STRING COMPARISON ROUTINE. DE+1 AND HL POINT TO
; STRINGS. B IS LENGTH. RETURNS ZERO SET IF TRUE MATCH.
;
TEST:   PUSH HL                         ; Save
        PUSH DE                         ; every-
        PUSH BC                         ; body.
        LD      A,(DE)                  ; Get string length in table.
        CP      B                       ; Is it same as string in line?
        JR      NZ,TOV                  ; If not, the test is over.
        INC     DE                      ; Point to first byte of string.
TLOOP:  LD      A,(DE)                  ; Get a byte from table.
        CP      (HL)                    ; Does it match so far?
        JR      NZ,TOV                  ; If not, the test is over.
        INC     HL                      ; Else look at next bytes to
        INC     DE                      ; compare, and
        DJNZ    TLOOP                   ; continue comparison.
TOV:    POP     BC                      ; TRUE comparison. Zero is set.
        POP     DE                      ; Restore
        POP     HL                      ; everybody, and
        RET                             ; return.
;
; ADD VALUE IN A TO DE REGISTER PAIR
;
ADE:    ADD     A,E
        LD      E,A
        LD      A,0
        ADC     A,D
        LD      D,A
        RET
;
; FULL CHECK...IF NO MORE SPACE, SEND MSG AND ABORT.
;
FULLCK: PUSH    HL                      ; Save HL
        LD      HL,LASTAD               ; Point to last possible address.
        LD      C,B                     ; BC will contain the length of
        LD      B,0                     ; characters we want to add.
        SBC     HL,BC                   ; Adjust HL by that length.
        LD      B,C                     ; Must preserve B register.
        SBC     HL,DE                   ; If space comp is not positive,
        JR      C,FULLUP                ; then it won't fit. Blow it off.
        POP     HL                      ; Else OK, restore HL and
        RET                             ; return.
FULLUP: LD      DE,NOWAY+OFF            ; Point to NO ROOM msg, and
        JP      SPILL+OFF               ; bail out of here.
;
IGNOR:
        LD      HL,(BUF0+OFF)           ; Point to start of input buffer,
```

```
         INC     DE                      ; bump the output buffer pointer,
         INC     HL                      ; look a name length byte in table.
         LD      C,(HL)                  ; put name length in C and
         LD      B,Ø                     ; zero B. BC contains name length.
         INC     HL                      ; Point to start of name.
         LDIR                            ; Copy name to output buffer.
         LD      A,':'                   ; We need a colon to follow
         LD      (DE),A                  ; the name and
         INC     DE                      ; after that
         LD      A,'='                   ; an equal sign
         LD      (DE),A                  ; in the output buffer.
         INC     DE                      ; Bump the output buffer pointer.
         LD      C,(HL)                  ; HL will point to string length,
         LD      B,Ø                     ; so put that into BC like before.
         INC     HL                      ; HL points to start of string.
         LDIR                            ; Move string definition to buffer.
         LD      A,CR                    ; Next put a carriage return
         LD      (DE),A                  ; into output buffer,
         INC     DE                      ; followed by
         LD      A,LF                    ; a line feed
         LD      (DE),A                  ; and one whole output line is done.
         INC     DE                      ; Bump the output buffer pointer,
         JR      LOOP                    ; and continue until table ends.
DONE:    LD      A,Ø1AH                  ; Seal off the buffer with an
         LD      (DE),A                  ; end-of-file character.
         EX      DE,HL                   ; HL points to the EOF mark.
         LD      DE,OBUF                 ; DE points to start of buffer.
         XOR     A                       ; Ensure carry not set,
         SBC     HL,DE                   ; and calculate length of buffer.
         RLC     L                       ; Simple means to divide the
         RL      H                       ; length by 128 to see how many
         INC     H                       ; sectors to write to disk.
         LD      B,H                     ; Put number of sectors in B.
WLOOP:   LD      C,26                    ; Set up DMA address.
         PUSH    BC                      ; Save B
         PUSH    DE                      ; Save where we are in buffer.
         CALL    CPMIO                   ; Establish DMA address.
         LD      DE,FCB                  ; Point to file control block.
         LD      C,21                    ; Set up WRITE SEQUENTIAL function,
         CALL    CPMIO                   ; and write 128 bytes to disk.
         POP     DE                      ; Retrieve our DMA address.
         POP     BC                      ; Retrieve our sector counter.
         OR      A                       ; Was write sucessful?
         JR      NZ,WIND                 ; If not, close file, and notify.
         LD      HL,8ØH                  ; Else update the DMA address
         ADD     HL,DE                   ; to point to next 128 byte block,
         EX      DE,HL                   ; which goes in DE,
         DJNZ    WLOOP                   ; and loop until sector count=Ø.
WIND:    LD      DE,FCB                  ; Point to file control block.
         LD      C,16                    ; Set up CLOSE function.
         JR      Z,FIN                   ; and if all is well, close and end.
         CALL    CPMIO                   ; else close and
         LD      DE,DFULL                ; point to error message.
BYE:     LD      C,9                     ; set up print function,
FIN:     CALL    CPMIO                   ; perform function,

         JP      ØØØØ                    ; and exit.
CERR:    DB      'FILENAME NOT SPECIFIED$'
NOTUP:   DB      'LNT NOT ACTIVE$'
DFULL:   DB      'DISK FULL OR R/O$'
OBUF:    DB      1AH
         END
```

```
        INC     HL                      ; then to length of buffer.
        XOR     A                       ; Load a zero there so the CCP
        LD      (HL),A                  ; wont see the input at all.
BYE:    LD      SP,(SAVE+OFF)           ; Give the CCP his stack back,
        RET                             ; and return control to the CCP.
;

LNT:    DB Ø                            ; Start of NAME TABLE.
ECODL:                                  ; End of code, for computing offset.
        ORG Ø1ØØØH                      ; A safe place to read in .LOG file.
LDMA:   DB 1AH                          ; A CP/M end-of-file mark.
        END
```

LISTING 3. SAVLOG program

```
; This is the SAVLOG program that is used to create an ASCII text file
; of all LOGICAL NAME DEFINITIONS that are currently active. The output
; file can be read by the LNT program to onload the entire set of defin-
; itions at once or modified using standard editors. The output file-
; name is required on the command line. An extention type of .LOG is
; recommanded but not mandatory. This program will assemble directly
; using Microsoft's M8Ø. Other assemblers may use DEFB and DEFW storage
; declarations instead and DB and DW.
;
        .Z8Ø
        ASEG
        ORG     1ØØH
CPMIO   EQU     ØØØ5                    ; CP/M entry point.
FCB     EQU     ØØ5CH                   ; Default file control block.
CR      EQU     13                      ; Carriage return.
LF      EQU     1Ø                      ; Line feed.
START:  LD      A,(ØØ8ØH)               ; See how long rest of line is.
        OR      A                       ; If not zero, then filename
        JR      NZ,CONT                 ; was specified. Go on..
        LD      DE,CERR                 ; Else you forgot. Point to msg,
        JP      BYE                     ; print it and abort.
CONT:   LD      DE,FCB                  ; Point to file control block.
        LD      C,19                    ; Set up delete function, and
        CALL    CPMIO                   ; zap the file if it exists.
        LD      DE,FCB                  ; Point to file control block.
        LD      C,22                    ; Set up MAKEFILE function, and
        CALL    CPMIO                   ; try to create it.
        CP      255                     ; If not bad return code, then
        JR      NZ,READY                ; continue. Jump to READY.
        LD      DE,DFULL                ; Else can't create file, notify
        JP      BYE                     ; and abort.
READY:  LD      HL,(ØØØ1)               ; Get location of BIOS+3.
        LD      DE,16Ø3H-1A2H           ; Calculate where CALL DOOR is
        XOR     A                       ; Ensure carry not set, and
        SBC     HL,DE                   ; HL should point to DOOR address.
        LD      A,5                     ; If a Ø5 is not found there, then
        CP      (HL)                    ; assume the LNT is active,
        JR      NZ,FOUND                ; and jump to FOUND.
        LD      DE,NOTUP                ; If it was Ø5, notify that LNT is
        JP      BYE                     ; not active and abort.
FOUND:  LD      E,(HL)                  ; Pick up DOOR location found at
        INC     HL                      ; HL and save in
        LD      D,(HL)                  ; DE register.
        DEC     DE                      ; Just before DOOR is strorage for
        LD      A,(DE)                  ; address of table. Pick up high
        LD      H,A                     ; byte and put into H and
        DEC     DE                      ; pick up low byte and
        LD      A,(DE)                  ; put into L, so that
        LD      L,A                     ; HL now points to table.
        LD      DE,OBUF                 ; Point DE to start of output buffer.
LOOP:   LD      A,(HL)                  ; Get a byte from the table.
        OR      A                       ; If you find a zero there,
        JR      Z,DONE                  ; then jump to DONE.
        LD      A,'#'                   ; Else not done, put a #
        LD      (DE),A                  ; into the output buffer,
```

20 Reasons to Buy dBASE III

by Scott L. Patashnick

To say that the name dBASE III is a little misleading is truly an understatement! There should be a law against mixing Silicon Valley and Madison Avenue... indeed, a much more conservative name like dBASE II, Version 3.0, would have been more appropriate... would have been more expected... and might not have conjured up expectations of the still not yet possible. But although dBASE III it's not, it is a super Version 3.0 with a number of very significant enhancements making the nominal upgrade charge (plus the extra memory, dBASE III requires at least 256K) or an outright purchase of the new version a great value for IBM/PC and IBM/XT owners.

I am assuming that most of you reading this article are upgrading you present version of dBASE III, so this won't be a dBASE Primer... or an all-encompassing evaluation of the new version; but rather my goal is to provide enough of an overview to allow users already committed to dBASE II to make a decision to upgrade or not based upon 20 improvements (out of hundreds that were made) selected by a fellow user (who doesn't even own any Ashton-Tate stock) instead of reading the hype of a Madison Avenue writer with less than an "arm's length" interest in pushing dBASE III.

As a final note before starting, I have included after each improvement a short explanation of how the new dBASE III feature might impact a specific application... I have chosen a hypothetical inventory application to use for those explanations. My perspective in selecting the following improvements is based upon that of an experienced program developer writing dBASE III programs that will be "user friendly" for others to use. There have been many other note worthy features added to dBASE III for the beginning programmer or those of you purchasing dBASE III for your own use that make dBASE III almost as easy to use as the Madison Avenue writer suggests.

File capacity

The maximum number of records per file has been increased to 1,000,000,000 (one billion) from the dBASE II limit of 65,535. Applications with files exceeding the new capacity might well be a candidate for a minicomputer. There should no longer be a reason for having to break up or chain database files together because of a capacity limitation. Programming can now be done faster... using a single large file is faster operationally... and now applications addressing large databases are practical candidates for use with dBASE III.

This means for an inventory program that had only 1,000 stock keeping units (SKU's) with a quantity of 100 of each SKU (item), the user may now keep track of each item by serial number (100 items x 1000 different items equals 100,000 records). This makes dBASE III practical for inventory applications of manufacturers of expensive, serialized products/items.

Fields per record

The maximum number of fields per record has been expanded to 128 fields (or descriptors) from the dBASE II limits of only 32 fields. You no longer have to spend extra programming time coding and then packing several descriptors into a single field (ie. a single field in dBASE II might be called CODE and contain 10 characters with each character having a special meaning; the data entry operator would have to know that the first character was the color code, the second character was the size code and so on). Data entry is faster and easier since each field (or descriptor) can now be separate; and 128 is a reasonable number of descriptors for many more applications that 32 was.

Our inventory application can now more easily keep track of many more descriptive elements about a particular inventory item. For example, a manufacturing inventory application could now keep an inventory of built sub-assemblies with the record having a field for each of the components contained in the sub-assembly. Perhaps even the serial number of each component; or the batch number the components were purchased under. This adds a new 'third dimension' for more creative dBASE III inventory solutions.

Files opened simultaneously

The maximum number of database files that may be open simultaneously at one time has been increased to 10, from the dBASE II limit of two. Combined with the previous enhancement of more fields per record, application development time can be dramatically reduced for advanced applications since the amount of coding required to either pack multiple descriptors into a single field or the code required for switching files back and forth between Primary and Secondary files has been eliminated. Plus dBASE III itself can run much faster when more (if not all) of the files called from an application remain open in memory for the duration of the program.

Inventory application designs using multiple files that were previously too slow to implement under dBASE II might now be feasible. A comprehensive inventory application might link purchase order, sales order entry and accounts payable modules with the inventory program. With all of these files accessible at one time, an operator query could interactively view not only the actual inventory on-hand for a specific item, but also see how many are on-order and how many are on-hold (allocated) for a pending order.

Use of RAM memory

The minimum required amount

of RAM memory needed has been increased to 256K from the dBASE II requirement of 64K for eight-bit systems and 128K for sixteen-bit systems. dBASE III uses the extra memory to dramatically reduce the number of overlay swaps in and out of memory making dBASE III itself operate much faster. Additionally, when using the equivalent of the dBASE II 'QUIT TO' command, there is no long delay since dBASE III isn't swapped out of memory; rather, the 'called program' is loaded into the remaining memory space available. dBASE III can now quickly execute COM or EXE files directly from the command level and then return control to dBASE III when done. Depending upon the size of the programs 'called' by dBASE III into memory, you may require additional memory above the minimum requirement of 256K.

The faster interaction with the 'called programs' would allow our inventory application to be capable of timely calling programs that might poll a cash register, or an OCR (optical character reader) scanner, or a hand-held data entry unit or another inventory related device that would pass inventory data to the system for processing by dBASE III.

Memory variables

The maximum number of memory variables has been expanded to 256 from the dBASE II limit of only 64. The amount of memory space has also been enlarged to accomodate the additional variables; a new limit of 6000 bytes compares to the dBASE II limit of 1536 bytes. More data can now be manipulated in memory using these temporary variables instead of swapping memory files in and out of the active user space.

The availability of more temporary memory variables allows our inventory program to buffer a greater amount of data entries (or changes) before writing to disk; for example a sales order for numerous items could be completely assembled in memory and then aborted at any time before acceptance with no consequence to the actual inventory data files.

Mathematical functions

The functions of square root, natural logarithm (base e) and the natural exponent (e) of a number have been added to dBASE III. These functions were not present in dBASE II. Applications previously requiring these mathematical functions (not present in dBASE II) can now be considered using dBASE III... plus having these functions directly accessible as a dBASE III command is much faster than utilizing other 'called programs' to perform these calculations.

More complex math capabilities can now be utilized by our inventory application; instead of manually setting minimum and maximum inventory guidelines, our system could now automatically set economic ordering quantity (EOQ) values and recommend orders to the user based upon the systems 'learning experience'.

Automatic range checking

A range option has been added which when used with the 'GET' or 'SAY' commands can limit numeric or date entries accepted by specific upper and lower values. This kind of data entry checking could only be duplicated in dBASE II with a command file of numerous steps.

Having a range check integrated into dBASE III as a command eliminates programming time to write such checks, increases the quality of the data entered and runs faster than comparable code written in dBASE II to perform the same function.

Application: This feature is generally used to set HI-LO (high and low range) values for performing automatic error checking during data entry of inventory quantities or costs. In our example, if we had no items priced over $10, we could eliminate a data entry error of a $9 item being entered as $90. And if we had no inventory items valued under $.10, we could also eliminate a data entry error of a $.90 item being entered as $.09.

Numeric precision

The numeric accuracy has been increased to 15.9 digits from 10 digits in dBASE II.

Flexibility. Applications requiring accuracy beyond the 10 digit capacity of dBASE II can now be addressed using dBASE III.

Application: Our inventory application can now be installed in user environments where the total inventory valuation exceeds the previous dBASE II limit of $100 million dollars.

New field types

Several new field types have been added to the dBASE II field types of N (numeric), C (character) and L (logical). The new M (memo) field type indicates a field size that is variable in size and is automatically stored in an auxillary file to the main database file (memo file type is .DBT). Space utilized by this field type ranges from 0 bytes (no entry, 0 characters) to a maximum field size of 4096 bytes (4096 characters). Each memo field occupies 10 bytes in the master database file (file type .DBF).

The memo field allows a variable length field to be attached to a record with a minimal overhead of only 10 bytes... yet when needed, up to 4096 characters are available. Now with virtually no additional programming, memo 'tags' can be attached to records for occasional comment or descriptive information that varies in length.

A memo field could be used in our inventory program to contain and explanation of a special use for that item. For example a memo field for an inventory item of diskettes could list what computer systems use that kind of disk. Another inventory application might store in the memo area the inventory number of related items usually purchased together with the select item.

A new procedure (subroutine) feature

Up to 32 command programs may now be stored together in a single file, compared to the dBASE II limit of only one command program per command file. You may now store up to 32 short 'subroutine-like' programs together in a single 'PROCEDURE' file. The file is structured like a normal command file except that each of the subroutines must begin with the command 'PROCEDURE <name>' and end with a 'RETURN' command. The subroutines stored in this single file may then be 'called' from an executing command file with the 'SET PROCEDURE' command. For larger dBASE III applications, this feature saves valuable directory space (which is limited by the operating system).

Our inventory application has the potential to re-use common subroutines over and over by different modules; for example a subroutine could be written to clear the screen and display a common main menu header throughout all modules. This 'header' might consist of a dotted line, followed on the next line by the program name and version number, followed on the next line by the current date and time, and finally followed by another dotted line. A variable could be imbedded in the subroutine to indicate the appropriate module name:

(sample subroutine to printer display)	
SAMPLE INVENTORY HEADER, Version 1.10	
Module Name Goes Here	
01/01/80	01:20:00

Printing labels

Mailing labels can now automatically be printed using a label format file storing the labels specifications including:

Label width	1 - 120 characters
Label height	1 - 16 lines
Lines between labels	1 - 16 lines
Spaces between labels	1 - 120 spaces
Number of labels across	1 - 52 across

This compares to no automatic label printing function under dBASE II at all.

The larger mailing list applications that lend themselves so well to a database management system like dBASE III, can now be fully addressed (no pun intended) with an equally dynamic label printing capability. Gone are the days of writing command files to manually read multiple records, buffer the variables into memory (if you had enough space), and then calculating the print positions across the page.

Incorporating multiple label printing routines into our inventory application can new be accomplished with minimal programming effort. Inventory information can now quickly and easily be printed on different size labels for applications such as pricing stickers on inventory receipts or shelf labels to mark inventory bins. Plus dBASE III will automatically adjust for up to 52 labels across!

New functions

The ability to erase information on the display screen has been expanded to include a clear to end-of-line (EOL) and clear to end-of-page (EOP) function compared to the dBASE II command of only clear screen (ERASE).

Using the 'h' command now allows you to specify a point on the screen and erase what is displayed on the screen from that point to the end of that particular line; or from that point to the bottom of the screen. Advanced custom written data entry routines can be now be tackled that will selectively erase only portions of the screen at a time.

User friendly screen routines can now be processed faster; for example the previously suggested subroutine (the new 'PROCEDURE' feature) used to print the 'header' on the screen could now print the 'header' only once when the program is first executed and then only the module name and time could be selectively erased and changed using the new commands. Utilizing these features, changes to the data displayed on the screen can be minimized since it is no longer necessary to re-display a complete new screen whenever only a small portion of the screen needs to be changed.

Interactive file linkage

Two files may now be addressed simultaneously in a master-slave relationship whereas a field selected in the master file becomes the index field in the slave file. The power of this new command could only be duplicated in dBASE II through a command file of numerous steps.

The command 'SET RELATION TO <variable> INTO <slave file with same variable name>' links two open database files based upon a field that is common to both. In dBASE II terms, this means that you can execute a single 'FIND' in the primary file and both the primary file and the secondary file will advance to their respective records. It is no longer necessary to do a 'FIND' in the primary file and then 'SELECT' the secondary file and perform another 'FIND' there.

Applying this feature to our inventory application, we could have an interactive sales order entry program with a database file called ORDERS.DBF and our inventory file called INVEN.DBF. Both of these files would have a common field called PART. Using the command 'SET RELATION TO PART INTO INVEN' we could simply 'FIND' and order number in the ORDERS.DBF and dBASE III would automatically position the record pointer in the INVEN.DBF file to a record having the same PART number as that contained in the ORDERS.DBF record.The full part description stored in the inventory file could now be displayed and would not have to be redundantly stored in the ORDERS.DBF file.

New date functions

A new D (date) field type has been added; plus 8 extra date related functions and a 9th time feature to provide command level options to convert numeric date entries into their respective alphabetic month or day-of-the-week equivalents. dBASE II only had a command to read the current date from the operating system and another command to change or set the date.

Cumbersome command files no longer have to be written to covert the '12' in '12/10/80' to the alphabetic equivalent of 'DECEMBER'. dBASE III can now perform this operation as a single command 'CMONTH(DATE())'. Additionally, a new data field type called 'D' (date) has been added for further providing more reliable data entry. Dates are valiated as entered and can be presented in either the American format (MM/DD/YY) or the European format (DD/MM/YY). Using this new date field type allows dates to be subtracted from one another (to calculate the number of days-between-dates) or a numeric value may be added or subtracted from a date variable to computer a new month, day and year. The following list briefly describes the new commands assuming '12/10/80' has been stored to DATE():

CDOW	Tuesday (calculates day of week)
CTOD	12/10/80 (character to date conversion)
CMONTH	December (calculates the month name)
DATE	12/10/83 (displays system date)
DAY	10 (displays day of month)
DOW	3 (displays numeric value for day; Sunday = 1)
DTOC	12/10/80 (date to character string conversion)
MONTH	12 (displays numeric value for month; Jan = 1)
YEAR	1980 (displays year expanded to 4 digits)
TIME	12:30:00 (displays system time)

The most important aspect of a 'user friendly' application is its ability to communicate with the user in the most unambiguous method available. To be able to quickly and easily display a date in the format 'Tuesday, December 10, 1980' is much more desirable and understandable by an operator than 12/10/80. Use of the date fields within our inventory application could be further applied by easily being able to computerize the age of inventory items by simply subtracting the date the inventory was received from the current date. Using dBASE III this task is accomplished by merely entering '? DATE() - RECVDATE' resulting in a numeric value indicating how many days that item has been on the shelf.

Modify structure command

The modify structure command now automatically renames the original file with the extention '.BAK' and allows the new structure to be changed. When all changes have been made, the data in the '.BAK' file is automatically appended to the new file. The modify structure command under dBASE II simple deleted the entire data file (without making any back-up first)!

This features eliminates the chance of accidently deleting a complete database. Additionally it now allows changes to the data structure to be made quickly and easily. Changing a field name in dBASE III takes only a few seconds, plus the time to append the back-up data file to the new structure. To change a field name in dBASE II was a major undertaking; first you had to copy the structure to a new file; then you had to change the field; then copy the original data file to an ASCII file using the SDF option; and finally append the data from the ASCII file to the new file again using the SDF option (plus you had to make sure all elements were in the correct physical order).

This feature means that future enhancements to our inventory application that require additional data files can be quickly and easily added without any chance of losing data. Of course, since this feature copies data from the old file to the new file you should determine in advance that there is sufficient disk space for the new file.

New 'set unique' command

This feature is 'set' prior to indexing a file and will prepare an ordered list of records without any duplicates. This feature could only be accomplished in dBASE II with a command file of numerous steps. This option can be used to either perform an error testing function by counting the number of unique records in a file (which when compared to the total number of records in the file will indicate whether or not duplicates exist), or, it can be used to prepare a list of all the unique descriptors entered in a particular data field.

The 'SET UNIQUE' command could be used to confirm that no duplicate inventory numbers have been entered into the system. A more practical use however would be in preparing reports. For example a report listing all of the different vendors from whom inventory had been purchased.

New text command

This feature is used in command files to simply display lines of text on the screen; or to print these lines of text on the printer. This compares with the rather awkward exercise of using the '?' or 'h' commands within dBASE II. To display a block of text can now be done from within a command file by simply entering the command 'TEXT' followed by as many lines of text as desired; and ultimately followed by the 'ENDTEXT' command. All of the text will ouput exactly as entered without being 'interpreted' by dBASE III.

Using the dBASE III 'TEXT' feature will allow us an easy way to implement an optional 'on-line' help function integrated into our inventory application. A test could be included in data entry modules looking for a 'HELP' request from the operator. When detected, a help selection could cause a screen of text to be displayed (perhaps 32 'pages' of instruction text might be stored in a single 'PROCEDURE' file). This would minimize the need for the labor intensive and costly preparation of a printed user's manual by making our appliation with a built-in tutor option.

New alias feature

The alias command allows files and field names to have a second, alternative name. A feature of this magnitude did not exist in dBASE II.

This feature allows files to be re-assigned a different name (or alias) within an application. The use of an alias is required by dBASE III when opening files; in fact, when more than one file is opened, dBASE III will automatically assign each file opened the respective alias names of A, B, C, D and so on (or you can set your own aliases). The use of a short alias is particularly useful in multi-file applications in labeling or identifying a file that a selected field is being called from. For example, if you opened the files DEMOFILE.DBF and TESTFILE.DBF; and both of these files had a field called ZIPCODE, the A-ZIPCODE would refer to the ZIPCODE from the DEMOFILE.DBF file, and B-ZIPCODE would refer to the ZIPCODE in the TESTFILE.DBF file.

Consistency in software is important, yet sometimes difficult to achieve with a limitation of a single file name. Using the alias command it is now feasible in our inventory program to have all of the command files designated by the first two letters INxxxx.PRG. As a matter of fact, all of the command, database, memory, label and index files used by our application should have the first two-letter prefix of 'IN'. This makes it easy to copy programs and display the inventory related modules on a disk using the directory 'wild-card' functions (ie. IN???.PRG, IN*.*). However, using a consistent name beginning with 'IN' can be extremely confusing when identifying what function a module performs. This is where an alias simplies documentation. Within the inventory programs, all of the files used can be referenced by aliases making the function (or contents) of these files self-explanatory.

The true power of this feature however, is in its ability to allow command files to be written that are independent of the variable names used with it... in other words, independent of the data files (and fields) used with it. In the case of our inventory application, this would allow command files written specifically for this application to be generically used for other totally different applications... with other data files (just referenced by the same aliases).

dBASE II conversion

The program dCONVERT allows dBASE II applications to be converted to operate under dBASE III... of course some applications may require some additional changes to run more efficiently, but dCONVERT makes the conversion very enticing. In addition to converting the .PRG files, dCONVERT also makes necessary changes to the .DBF, .MEM and other file types to reflect and utilize some of the enhancements of dBASE III.

I have only high praise for Ashton-Tate's decision to try to make the command set of dBASE III a super-set of dBASE II... thus allowing dBASE programs to operate in the dBASE III environment. This allows thousands of dBASE II programs to be relatively easily transported to dBASE III; however, I believe most of these conversions will only be a temporary step, since the enhancements described in this article will have a far reaching impact upon the very logic and system design aspects of using dBASE III to solve problems.

Use of the conversion utility provided with dBASE III will allow us to immediately implement our existing inventory application (already developed under dBASE II). This will relatively quickly allow us to begin operating in the improved, faster and more powerful dBASE III setting. Going the other way, dCONVERT will also convert dBASE III files (which meet the dBASE II limitations) to a dBASE II readable format.

dBASE III specifications

Based upon the previous explanations and examples of dBASE III, the following table should be understandable:

dBASE III Files:

Number of records	1 billion (maximum)
Number of bytes	2 billion (maximum)
Record size	4000 bytes per file 512 kilobytes in memo file
Fields	128 bytes (maximum)
Field Sizes:	
Character fields	254 bytes (maximum)
Date fields	8 bytes (maximum)
Logical fields	1 byte (maximum)
Memo fields	4096 bytes (maximum)
Numeric fields	19 bytes (maximum)

File Operations:

15 files can be open simultaneously (combined maximum)

10 database files can be open at one time (maximum). A database file counts as two when memo fields are used.

7 index files may be open per database file.

1 format file may be open per database file.

Numeric Accuracy:

15.9 digits (the decimal point does not count in determining the numeric accuracy.

Memory Variables:

256 is the maximum number of active memory variables that may be used at one time.

6000 bytes of memory are reserved for active memory variables.

Conclusion

In conclusion I almost want to thank Ashton-Tate for making their initial releases of dBASE II so limited... when I reflect on what I have done and what other far more experienced programmers have accomplished, in spite of those limitations... plus the fact Ashton-Tate has made dBASE II a sub-set of dBASE III... and the increased capabilities of dBASE III... well, the tools of the programmer are improving... bit by bit. I have to agree with the comments of Adam Greene (which are included with the dBASE III manual) in which he states "dBASE III is now the most mature, best performing, business programming language available on any computer."

About the author

Scott Patashnick designs and directs computer system development for EPCO Systems Ltd., Inc., his family-owned group of New England based companies. He has authored several dBASE II application packages for small business and currently is installing a dBASE Sales Tracking and Officer Incentive System for banking clients with locations in the Northeast.

16 Bit Lisp and Prolog Implementations

by William G. Wong

Prolog and Lisp are the two main programming languages currently being used for artificial intelligence (AI) projects such as machine learning, robotics, and problem solving. Most AI work using these languages has been done on large mainframes due to memory requirements of the applications and the language implementations. A number of Lisp (see "LISPing in Numbers", Microsystems, Aug. 1983 and "Three More Lisps for CP/M", Microsystems, Dec. 1983) and Prolog (see "Prolog", Microsystems, Jan. 1984) implementations are available under CP/M V2.2 but these tend to be restricted by the 64 kbyte memory limitation of the 8-bit processors. However, there are now a number of 16-bit Lisp and Prolog implementations which take advantage of the larger address space (1 megabyte) of the INTEL 8088/8086 or of new high-speed processors such as the 8 Mhz 80186).

The new language implementations also provide more flexible and sophisticated tools including resident screen editors, graphics support, and floating point hardware (8087) support. All this makes the new Lisp and Prolog implementations on micros practical development tools for AI projects, not just experimental toys.

The three Lisps being reviewed are IQ Lisp from Integral Quality, muLisp-82 from the Soft Warehouse, and TLC-Lisp from The Lisp Company (TLC). Also being covered is a version of Prolog called micro-Prolog from Logic Programming Associates, Ltd. Prolog is being reviewed with the Lisp implementations because of the similarity in operation, implementation and application. These systems were tested on the 8088-based Chameleon Plus (IBM PC compatible unit) with 256K of RAM and graphics support. TLC-Lisp, muLisp-82, and micro-Prolog are also available for 8-bit CP/M systems which have been reviewed in previous articles. The 8 bit implementations are upward compatible with their 16-bit counterparts.

Documentation

MuLisp-82. The muLisp-82 documentation has seen a slight improvement over its 8 bit counterpart. Microsoft (distributor for muLisp) has added the more impressive green and white binder containing a professional typeset reference manual. information relating to the 16 bit version has been added and various corrections and updates have also been included. The table of contents and indices are complete and correct.

Still, the reference manual retains some of the bad points found in the older 8-bit documentation. Both lack a sufficient number of concise examples to offset the terse and sometimes incomplete function descriptions. When describing functions, the muLisp-82 manual uses the Backus-Naur Form (BNF), which causes confusion. For example, listing 1 shows the descriptive syntax for the Lisp EQUAL function, using both BNF and Lisp. I prefer the latter for clarity and consistency.

PROGRAM #1

```
BNF Syntax (used in muLisp documentation)

   CBV EQUAL [X, Y] :=
       ATOM [X] --> EQ [X, Y];,
       ATOM [Y] --> NIL;,
       EQUAL [CAR [X], CAR [Y]] -->
           EQUAL [CDR [X], CDR [Y]];,
       NIL;

Lisp Syntax

   (DEFUN EQUAL
         (LAMBDA (X Y)
                 (COND ((ATOM X)  (EQ X Y))
                       ((ATOM Y)  NIL)
                       ((EQUAL (CAR X) (CAR Y))
                        (EQUAL (CDR X) (CDR Y)) )
                       (NIL) ) ) )
```

As it turns out, the Lisp syntax description is also a valid Lisp definition for the function and serves as an example of the language. Maybe the next version of the documentation will be easier to read. Even so, the current edition is usable.

IQ Lisp IQ Lisp has no 8-bit counterpart. The manual (one of the best) is packaged in a nice blue and white, three- ring binder, and includes a great deal of information about the system. The information is well organized and well presented, with concise examples for each function definition. The section on internal system structure and assembly language interface is the best I have seen. It even includes a complete example, with corresponding files on the distribution diskette.

The only faults are minor. The manual is divided into four chapters; a table of contents and index are associated with each chapter, but there are none for the entire document. Finding a particular function definition is sometimes difficult unless you know which chapter contains its description. A complete functional and annnotated index would solve this problem. Online documentation is limited to a list of function parameters.

TLC Lisp The TLC-Lisp documentation is currently being revised and comments pertain to the new version which has just been completed. It is similar to the **Thinking About TLC-Logo** which is excellent. As with IQ Lisp, the amount of documentation is substantial because of the large number of available functions which must all be described. It is well presented and organized. Unfortunately, no online documentation is currently supplied.

Micro-Prolog. The new micro-Prolog manuals are essentially an updated version of the 8-bit reference manual and primer which were very good. The best improvement is in the packaging. The manuals are now contained in small PC-style ring binders instead bound paperback books, which tended to become tattered due to heavy use. Unfortunately, a four-ring binder is used, so conventional three-ring plastic floppy disk inserts do not work too well. However, the excellent organization and content of the primer and reference manual make such a minor problem seem trivial.

This is the only package which can be used by itself to learn the language (Prolog). Exercises and complete answers are included in the primer, which can be used as a basis for a course on Prolog in general. The reference manual uses the same format. The table of contents and index are complete and informative.

None of the packages lacks any essential information. It is just a matter of packaging and presentation. The micro-Prolog and TLC-Lisp manuals are very good; the others are adequate.

Compatibility and Enhancements

muLisp. muLisp runs under PC-DOS, MS-DOS, MP/M, CP/M, CP/M-86 and Concurrent CP/M-86. The 8- and 16-bit versions are compatible. This means that programs can be moved among a number of different microcomputers that can run muLisp.

muLisp-82 is essentially the same as the 8-bit version, so all comments regarding that implementation hold for the 16-bit version also. It does not really match any mainframe implementation but is close to the original Lisp 1.5 specification. Its quirks in handling conditionals, unbound atoms, and error conditions make transporting programs to other implementations difficult at best. It does not support macro's, floating point, or data structures such as vectors, but muLisp-82 is complete enough to handle many applications.

The muLisp-82 implementation does not include list-mapping functions or the PROG function (which is a kludge anyway), but does support a flexible multiple- exit LOOP function. A compatible operation can be supported on other Lisp systems using a Macro. Also, muLisp-82 does not check the number of parameters during a function call; this deficiency can cause problems, especially in debugging.

IQ Lisp. IQ Lisp runs only on PC-DOS (V1.1 or V2.0), but in view of the vast number of PC compatible machines this is not really a restriction. There is no 8-bit version. IQ Lisp contains a plethora of features, its basis matching MACLISP, a mainframe implementation. Programs can be moved easily between systems, provided that they use only a common set of operations that excludes things such as the graphics support in IQ Lisp. MACRO, mapping functions, floating point and vector support are included along with sophisticated error handling. Extensions include primitive graphics support, multiple window text support and an assembly language interface. These tend to be unique to IQ Lisp in terms of syntax and operation.

IQ Lisp supports the PROG function but allows you to hide its presence with a number of MACROs for creation of loop functions. The MACROs are recommended, and provide a much cleaner program interface, especially if programs are to be moved to other machines. Like muLisp-82, IQ Lisp does not check the number of function parameters during a function call.

IQ Lisp includes a package support facility found on many mainframe Lisp implementations. This feature is very useful in keeping functions grouped together and organized when developing a system. However, it does not support separate name spaces. A separate name space is useful because it allows each module to be kept apart from modules in other name spaces. Suppose, for example, that there is a function FOO in the data base search package and another function FOO in the screen manipulation package which are different. We would like to use both packages with our program. No changes are required if separate name spaces are available because each FOO function is distinct within its own name space. But in IQ Lisp, you can only have one FOO so one of the FOOs must be renamed to something which does not cause a conflict. Unfortunately, the renaming requires modifying the corresponding package; this can be difficult.

TLC-Lisp. TLC-Lisp has an 8 bit sibling and runs on CP/M, MP/M, CP/M-86 and Concurrent CP/M-86. It is based on MACLISP but has a number of extensions. MACROs, mapping functions, vectors, floating point numbers and error handling support are included. Extensions include object support, a p-code assembler and LOGO-style turtle graphics.

TLC-Lisp does not include the PROG function. Instead, there is a multiple-exit DO loop function. In addition, it supports tail-recursion optimization, which means that loops can be written as functions which call themselves. This optimization actually provides a more general solution to the problem in a cleaner fashion. For example, the following function could be used to print the number 1 forever (not practical but a simple example).

```
(DE PRINT-1-FOREVER ()
        (PRINT 1)
        (PRINT-1-FOREVER) )
```

Running this function on IQ Lisp or muLisp-82 would cause the system to run out of stack space after some period of time. It would run forever under TLC-Lisp and micro-Prolog (which also has this optimization), and the function would use only one level within the program evaluation stack.

TLC-Lisp supports true modularized programming by providing separate name spaces. Each space is named and the routines for entering information and printing it out can be setup to handle the different name spaces. For example, the data base search space may be named DBMS (for lack of a better name) and the floating point package space could be called FP. References to the DBMS function FOO would be written as DBMS:FOO and likewise the FP function would be FP:FOO. The references within each space are really to FOO but only to the one within the space.

micro-Prolog. Prolog is a newer language than Lisp and there are fewer mainframe implementations but micro-Prolog matches these in terms of operation. However, it tends to differ in terms of syntax. The internal program storage uses a conventional Lisp list scheme for consistency. The internal form can be used directly or manipulated through various front-end programs that are supplied with the system to translate the list syntax into a more English-like syntax. For example:

Internal Lisp Syntax

```
((X is-the-grandparent-of Z)
  (X is-the-parent-of Y)
  (Y is-the-parent-of Z))
```

SIMPLE Front-end English-like Syntax

```
X is-the-grandparent-of Z if X is-the-parent-of Y
                          andY is-the-parent-of Z
```

Transporting Prolog programs between different implementations tends to be easier since translation and pattern matching are fairly easy in Prolog. Also, the basic Horn clause structure of Prolog is retained by micro-Prolog.

Basic features include mapping functions, floating point support and error handling. Extensions include separate name space module support and tail-recursion optimization. The PC-DOS version essentailly matches the 8-bit CP/M implementation.

micro-Prolog also supports separate name spaces for module implementation. Like TLC-Lisp, the names of the functions are exported and imported in a clean fashion and local function definitions do not conflict with those in other packages. This feature actually makes micro-Prolog more powerful than some mainframe-based Prolog implementations.

Numbers, Strings, and Objects

muLisp-82. All four implementations contain built-in list support. Differences between the various implementations occur with other primitive objects like numbers and strings. For example, muLisp-82 supports atom names (strings) and infinite precision integers (not really infinite but close to it). Infinite precision integers are like variable length strings. The higher the value, the longer the string. Integers can have up to 600 decimal digits with muLisp. The character string support is fairly limited but the infinite precision integers is good.

IQ Lisp. IQ Lisp supports a number of different types of numbers and objects. The IQ Lisp numbers include small integers (less than 32767), infinite precision integers (up to 77,000 decimal digits!!!), short floating point numbers (4 bytes), and long floating point numbers (8 bytes). The floating point is supported both in software and on the 8087 numeric coprocessor (if this is available). Standard numeric functions are available plus floating point manipulation functions. Basic trigonometric functions are also supported.

As this were not enough, IQ Lisp also supports strings up to 32K in length with good string manipulation functions. Multidimensional arrays are also included. Array types include byte, small integers, small floating point numbers, large floating point numbers, and pointers. This variety allows a programmer to optimize access and storage requirements. Arrays are dynamically allocated and collected like lists.

IQ Lisp also supports special objects such as files and windows for I/O. Placing I/O support in objects greatly simplifies programming chores and leads to more structured programs.

TLC-Lisp. TLC-Lisp is on par with IQ Lisp in terms of objects. It supports small integers (less than 1024), and large integers (31 bits). It does not currently support infinite precision integers. Floating point support is restricted to 4 bytes but 8087 support is automatically detected. String manipulation is superb; it includes character insert and delete functions, along with a set of search functions. The insert and delete functions actually shift characters around to accomodate the changes. Character replacement in a string is also possible. The functions are used by the screen editor which is written in Lisp.

Only single dimension vectors are explicityly supported, but multidimension arrays could be supported by defining the appropriate functions. The advantage is that the additional multidimension support is not required if not used. Also, heterogeneous arrays are possible using this approach. TLC-Lisp also includes a number of vector manipulation functions not found in other implementations. This includes vector element insert, delete, and mapping functions.

The number of system objects provided in TLC-Lisp is too large to list, but includes I/O file objects, stream file objects, windows and turtles.

Micro-Prolog. micro-Prolog includes integers (less than 32767) and 6-byte decimal floating point numbers less than 10 to the 127th power. Numeric operations are limited to the basic arithmetic functions. Trigonometric or logarithmic functions must be defined by the user. Strings are in the form of atom names and are limited to 60 characters. Only string comparison and conversion to a list of characters are supported.

micro-Prolog structures are built from lists, so no special objects exist within the system. Files are referenced by name through special file support functions. Arrays are not supported either.

I/O Support

muLisp-82. Like its 8 bit counterpart, muLisp-82 provides limited file support. Only one input source and one output sink may be selected at any one time. These can be the console, the printer or a file. This feature allows loading and saving information via a disk file and interacting with the screen, but is not suitable for any disk file oriented operations. Console support is limited to operations which can be performed by using escape sequences.

IQ Lisp. IQ Lisp is much better with regard to console and file support. Multiple files can be opened at one time but only sequential operations are available. Random I/O must wait until the next release. However, sufficient functions are available to build a file application oriented to serial transactions.

IQ Lisp supports multiple, scrollable display windows on a single screen. The windows may overlap and will not obscure other windows. Windows are considered as files for the purposes of I/O operations. Window manipulation functions are provided to scroll a screen or change a screen's attributes, size or location. Moving a window logically does not move the contents on the display. Instead, subsequent operations are performed with regard to the new settings.

TLC-Lisp. TLC-Lisp supports multiple files and streams. Disk file access can be sequential or random; thus TLC-Lisp has the most flexible file manipulation system of the four tested. Random access is on a byte basis, and file update (read/write files) is possible. The stream-oriented files are very powerful, since they act as UNIX-style pipes. TLC-Lisp allows a function to insert information into a stream and be notified when more information is required. The function removing information from a stream does not know if the source is from a file, a string, or a generating function.

Windows are supported in two fashions. One is the turtle mode, in which there are two windows on the screen. The top one is the graphics drawing area for the turtle; the lower window is the interaction window where text is entered and displayed. The boundary between the two windows is adjustable. The graphics window is overlaid by the text window, but any drawing done by the turtle is retained if it occurs in the obscured partition. Reducing the size of the text window will show this retained information. This type of operation is convenient for testing, since the graphic operation of the functions can be shown in the top window and the status of the operation is shown in the bottom window. The final graphic result can be viewed by making the text window smaller or deleting it completely.

The second mode of operation is similar to that of the IQ Lisp windows, but is more primitive. Functions are available for scrolling regions of the screen and displaying characters within a region, but the user must write his own functions to build a more general system. This could be done using the stream objects.

micro-Prolog. micro-Prolog does not support multiple window operations but does allow multiple files to be used at one time. Both sequential and random I/O are supported. Unfortunately, files cannot be used in an update mode. Files must be opened for read or write access, but not both. Random file indexing is a holdover from CP/M. A file position is a list whose first element is the block number (128 bytes/block) and whose tail is the byte index within a block. For example, (2 | 55) refers to the 55th byte in block number 2 (both are base 0) which corresponds to byte 301 in the file. Luckily, as shown in listing #2, it is very easy to define new functions which translate byte references to the (block | index) form.

micro-Prolog also provides a limited number of format I/O operations too. This is primarily for fixed record size field access but can also be used for column aligned console and printer output.

Graphics Support

muLisp-82 and micro-Prolog have no inherent graphics capabilitIES except those which may be accessible through console escape sequences. The lack of graphics support is not important to the language unless, of course, you need graphics.

IQ Lisp provides limited color graphics support. Colors can be selected and drawing functions include POINT and LINE. These functions can reference a particular window, too. The LINE function requires both endpoints as parameters. Circles and more complex figures must be drawn using these basic functions.

TLC-Lisp provides LOGO-style "turtle graphics". This differs from the IQ Lisp mode of operation in that lines are drawn using the graphics cursor or "turtle" as its basis. The turtle has a location and direction. To draw a line, you give the turtle a distance to travel. For example, to draw a square you would use the following expression.

```
(REPEAT 4 (FD 100) (TR 90))
```

(FD 100) moves the turtle 100 units in the current direction drawing a line as it goes. (TR 90) turns the turtle right 90 degrees. Doing this 4 times gives a square. The similarity to Logo is a definite plus.

TLC-Lisp supports multiple turtles and you can feed all turtles the same operation at one time, which means you could program some pretty sophisticated games.

Resident Editors

Each system comes with a line-oriented structure editor or screen resident editor. A screen editor is preferable since it allows you to see changes immediately. The best thing about all the editor implementations is that they are written in the respective systems language. This means user modifications and enhancements are possible.

muLisp-82. The muStar system which comes with muLisp-82 is actually a menu-driven front end system that includes a screen-oriented text editor. The screen editor is adequate, but can be used only as a Lisp function editor. It is not capable of general text editing although it could be modified for such operation. The main drawback of the editor is that all commands must be memorized, since on-line help is available while you are using the editor. Also, saving a modified function can be a problem if the

LISTING #2

```
((TRANSLATE x ( y | z ))
        (INT x)              ; make sure x is an integer
        (TIMES y1 128 x)     ; get number of blocks
        (INT y1 y)           ; convert to integer
        (TIMES y2 128 y)     ; get number of integral blocks
        (SUM y2 z x) )       ; get byte index in block
 ((TRANSLATE x ( y | z ))
        (INT y)              ; make sure block number is an integer
        (INT z)              ; make sure block index is an integer
        (TIMES y 128 y1)     ; get block byte index
        (SUM y1 z x) )       ; add in block byte offset
```

syntax is incorrect, since muStar evaluates the function definition. An error in evaluation simply leaves you back in the editor with no indication of what problem was encountered. Even so, muStar has a definite edge over line oriented structure editors.

IQ Lisp. IQ Lisp comes with a line-oriented structure editor. Although not as easy to use as a screen-oriented editor, its complement of manipulation functions is good enough to meet most development projects. The advantage of the line-oriented structure editor is that a syntax check is done at the time a command is entered, so that the list being modified is always structured properly.

TLC-Lisp. TLC-Lisp initially had a Wordstar-like screen-oriented editor which is modeled after WordStar. Both were capable of editing list structures, such as function definitions, as well as normal text files. Although the editor is not as fully functional as a word processor the TLC-Lisp editor does serve as a good multipurpose editor. It is menu-driven but the menu consists of a single line at the top of the screen. With a little work, it could be used as a word processor so you could stay in Lisp to do all your work.

micro-Prolog. The editor which comes with micro-Prolog is a primitive line-oriented structure editor. It is sufficient for function definition modification but you will probably wind up using an external text editor to do program development.

Error Handling

muLisp-82. Error handling for muLisp-82 is the same as in the 8-bit version, which is not very good. In fact, it is sometimes hard to determine when an error has occurred because of the way muLisp handles errors. For example, normally the first atom of a list which is being evaluated is the name of a function definition. An error should be signalled if there is no definition. Instead, muLisp simply returns the list being evaluated as the result. Similar results occurs with file I/O errors and arithmetic errors. The tradeoff in these cases is speed - muLisp runs fast. But you must be careful when writing a program; errors can easily go undetected because a function will always return a value even if an error occurs. The basic trace functions can help.

IQ Lisp. IQ Lisp has the edge on error handling especially when the development support system is loaded. In addition to TRACE and BREAK functions, IQ Lisp also has the CATCH/THROW function pair. This is a useful programming construct that can also be used to recover from system errors. The ERROR function is also called when an error occurs and this function can be defined by the user. IQ Lisp also provides access to the system stack through built-in functions.

The high point is the window-oriented error monitor. This is invoked when an error is encountered that is not being handled by the user. In this case, the screen is divided into two windows. The larger top window is the stack control window, which contains the function being evaluated that has caused the error, along with other useful information. The lower window is the command window which can be used to modify values in the top window. In fact, any function may be invoked from the command window, thereby allowing you to change or examine values and function definitions.

The error monitor also makes good use of the IBM PC keyboard. For example, the arrow keys on the numeric keypad are used to browse up or down through the system stack so you can see what functions got the system into its current state. You can also back up and return a value from a function which was evaluated before the one in error. Very few systems provide debugging environment even close to that of IQ Lisp.

TLC-Lisp. TLC-Lisp does not have any equivalent of the error monitor found in IQ Lisp, but it does have all the other features such as CATCH/THROW, TRACE, and BREAK. TLC-Lisp also provides access to the program stack.

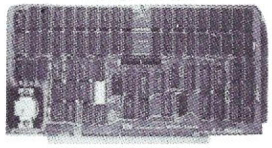
micro-Prolog. micro-Prolog provides an ERROR function which is called when an error occurs. Unfortunately, the two only choices at that point are to continue computation or abort it completely. However, tracing and break facilities are also provided.

TO BE CONTINUED

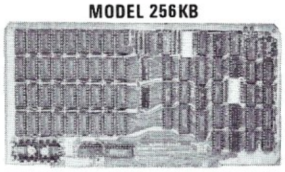
The second, and concluding part, of this Lisp review will appear in the next issue of Micro/Systems Journal. It will discuss such things as: garbage collection, assembly language interfacing and present the benchmark programs used to test these Lisp packages.

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

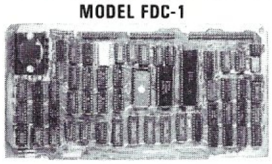
MODEL 256KM



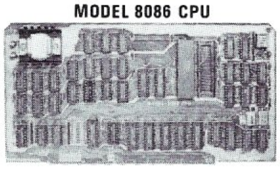
MODEL 256KB



MODEL FDC-1



MODEL 8086 CPU



256 K/1M DYNAMIC RAM Board Model 256KM $995
● 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 $395
● 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 $425
● 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 $425
● 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 1.2mbyte storage each, with CPM 86 operating system (Concurrent Dos available soon). Cabinet has room for full size 5¼" hard disk.

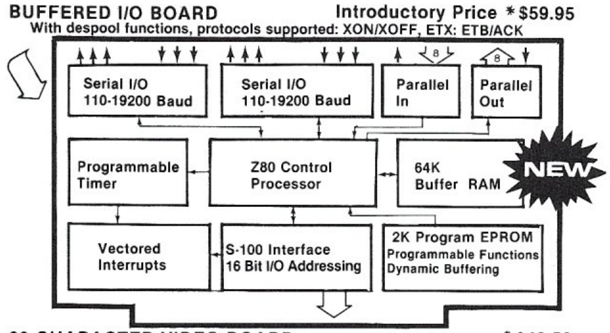
Operating Systems available: CPM 2.2, CPM 3.0, CPM 86, MSDOS.

*CPM is registered trade mark of Digital Research Inc. Z-100 is registered trade mark of Zeith Corporation.

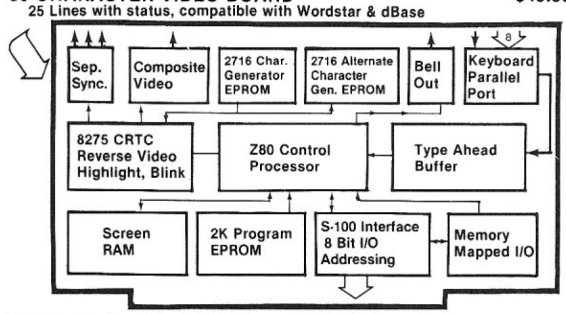
Please call for latest prices.
Prices subject to change without notice.

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

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

dataCURE

Protecting your valued diskettes

by Bruce Ratoff

Suppose you're doing some housecleaning on your word processing diskette when . . . oops! You deleted one file too many and wiped out WordMangler (your favorite word processor). No problem, you think. You'll just go fetch the master copy down from the shelf and be back in business faster than you can say "PIP". You pop the master diskette into your drive, try to log it in and much to your chagrin, you see:

```
BDOS error on B: bad sector
```

Who among us hasn't at one time or another had a day ruined by the appearance of this infamous bit of CP/M-ese? Or let's say that just as you are about to remove some one-of-a-kind diskette from your drive, your pet dachshund decides to jump into your lap, simultaneously knocking the computer's plug out of the wall. You keep your fingers crossed as you plug things back in and reboot, but once again, CP/M informs you that the disk in question has been rendered unpalatable.

These are but two of many possible ways that you can loose a valuable file, with the accompanying loss of time and money to replace or reconstruct it. Now, everyone knows that there's no substitute for keeping backup copies of critical data, but what do you do when your backup goes bad? A firm called Colorado Online believes they have found one possible answer in their software product, called "dataCURE."

What is dataCURE?

dataCURE is a method of protecting your valuable diskettes, so that if part of the diskette's contents is destroyed, it may be reconstructed using the part that is left. A disk that has been protected by dataCURE contains additional data that allows the program to recover the data that has been lost. dataCURE will recover files that have been lost by overwriting, as well as data lost due to hard media errors. Recovery is possible even if the directory area has been damaged.

To protect a disk with dataCURE, one must first start with a freshly-formatted blank diskette. The disk is then processed through dataCURE's initialization procedure, which reserves space for the protection files needed by dataCURE to perform its recovery functions. The protection files occupy about 5% of the total diskette space. Once the diskette has been initialized, you may then use PIP or some other file copy utility to add the files to be protected. You then run dataCURE's "protect" function, which constructs the actual protection tables and stores them in the previously allocated space. The diskette is now protected and may be stored away until needed.

When you need to use the protected disk, dataCURE's "verify" function will allow you to verify that no data has been modified since the "protect" function was last executed. If an error is discovered, you may then choose one of several available methods to recover the lost data. Recovery may be performed on the original erroneous disk, or you may use dataCURE to copy the defective diskette onto a good working diskette, and then perform the recovery steps there, without affecting the original.

Using dataCURE

dataCURE is delivered on a single diskette, along with a loose-leaf notebook containing the documentation. The notebook is divided into sections entitled "Release Notes", "Contents," "Introduction," "Tutorial," "Reference," and "General Notes." The Introduction gives general information on the program's capabilities and advantages. The Tutorial walks the user through installation and checkout of dataCURE on a particular system. During the course of this checkout, the user will verify the release diskette (which come dataCURE protected, of course) and create and recover from a variety fo typical errors. The Reference section is exactly that: a reference guide to the program, indexed by command name. General Notes gives some additional hints on protection, recovery and good working habits. The Release Notes section gives you a place to store the occasional update notices sent out by Colorado Online to its registered users. The documentation is attractively packaged and arranged, well organized, and complete without being cumbersome. All in all, an excellent job.

The program may be run in several ways. The simplest is to simply invoke it with no arguments by typing the command "DC" at the CP/M system prompt. dataCURE then loads and presents you with the following menu:

```
1 - HELP          5 - CORRECT
2 - INITIALIZE    6 - REPLACE
3 - PROTECT       7 - DUPLICATE
4 - VERIFY
```

Functions may then be selected from this menu by typing either the function number or the first letter of the function name. The user is then prompted for which drive to use and whatever other information is required by the particular command. Whenever the drive selected is the same as the dataCURE was loaded from, the user is given the opportunity to swap diskettes before the requested function is actually performed. This makes it possible to access all dataCURE features in a single-drive environment.

Once the user gains familiarity with the program, the menus may be skipped by entering the drive and function desired on the CP/M command line. For example, "DC B' /I" would mean to initialize the diskette in drive B:. This also makes it possible to put dataCURE commands into a SUBMIT file.

The seven dataCURE commands perform the following functions:

HELP: Puts up a one-page summary of commands and functions.

INITIALIZE: Readies a blank diskette for use by reserving the space needed to store dataCURE's protection tables.

PROTECT: Reads all active data on the diskette and fills in the protection tables to reflect the diskette's present contents.

VERIFY: Compares the diskette contents to the protection tables, verifying that the diskette contents have not changed since the last PROTECT function was executed.

CORRECT: Scans the diskette for errors and attempts to re-write any erroneous disk sectors in their original place on the disk. This is the normal way of fixing errors where no hard media errors exist.

REPLACE: Scans for errors and corrects them by moving the affected data groups to a different area of the disk. The erroneous groups are then locked out by allocating them to a dummy file called the "junkyard." This is one method of recovering a diskette that contains several "hard" media errors.

DUPLICATE: This function will copy the complete track-for-track contents of one diskette to another. Sectors containing hard media errors are bypassed. This creates a disk containing only "soft" errors, which may then be recovered using the CORRECT function. Another advantage of this feature is that the original diskette remains unaltered, since all recovery work is done on the copy.

How well does it work?

During the course of my testing, I tried a number of means of creating disk errors. These included using a sector-patch program to dliberately wipe out the directory, parts of the protection tables, and other random parts of the disk. I also tried running a magnet over part of my test disk. In the latter case, the disk could no longer be logged in by CP/M. After copying the diskette with the DUPLICATE function to eliminate the hard errors, dataCURE had no trouble recovering the lost data.

In most test cases, the program performed flawlessly, recovering all missing data. The only time it failed was when massive portions of the data had been destroyed. The author claims that the recovery techniques employed are tailored to the most commonly occuring disk errors, those affecting a small group of neighboring tracks and sectors. The claimed theoretical maximum number of bad bytes recoverable on an 8″ double-sided double-density diskette is about 8K. The protection tables are stored redundantly on the innermost and outermost protions of the diskette, so that if an error occurs inside the protection table, the alternate copy may be read to recover the protection information.

Conclusions

dataCURE is a well-written, well-documented product that provides an additional measure of security for your critical diskettes. It does not eliminate the need for backup copies, but it further ensures that those backups will be readable if needed. You may even find it use ful to dataCURE-protect your day-to-day working diskettes at the end of a lengthy session. In this way, you may not need to resort to your backup copy for most errors. Software vendors would also do well to consider adopting a protection method such as dataCURE. This would increase the chance of a release diskette surviving its travels from the software manufacturer to the end user.

dataCURE is priced at $104 (price includes shipping) for 8″ disks, or $99 for New Jersey residents. It is also available on popular $5^{1}/_{4}$″ formats for $114. For further information, contact:

Colorado Online
40 Balfour Lane
Ramsey, NJ 07446
(201) 327-5155

THE BOOK MART

The following selected books can be ordered thru Micro/Systems Journal. We furnish these titles only because we feel they are very worthwhile publications.

Interfacing to S-100/IEEE-696 Microcomputers by Sol Libes and Mark Garetz. The definitive book on the subject. Also has great application to hardware interfacing in general.
$17.95 (US. Can. Mex)
$27.95 (other foreign)

The 8086 Book by Russell Rector & George Alexy. Probably the most comprehensive reference book on both software and hardware for the 8086 and 8088. Packed with lots of program and circuit examples.
$19.95 (US. Can. Mex)
$29.95 (other foreign)

68000 Assembly Language Programming by Gerry Kane, Doug Hawkins & Lance Leventhal. A reference and how-to book with a wealth of fully debugged programming examples. Includes assembler conventions, I/O device programming and interfacing.
$19.95 (US. Can. Mex)
$29.95 (other foreign)

The Programmer's CP/M Handbook by Andy Johnson-Laird. A comprehensive reference on CP/M's internals, file system, CCP, BDOS BIOS, etc. Examples of CP/M customizing and utilities written in C are given.
$22.95 (US. Can. Mex)
$32.95 (other foreign)

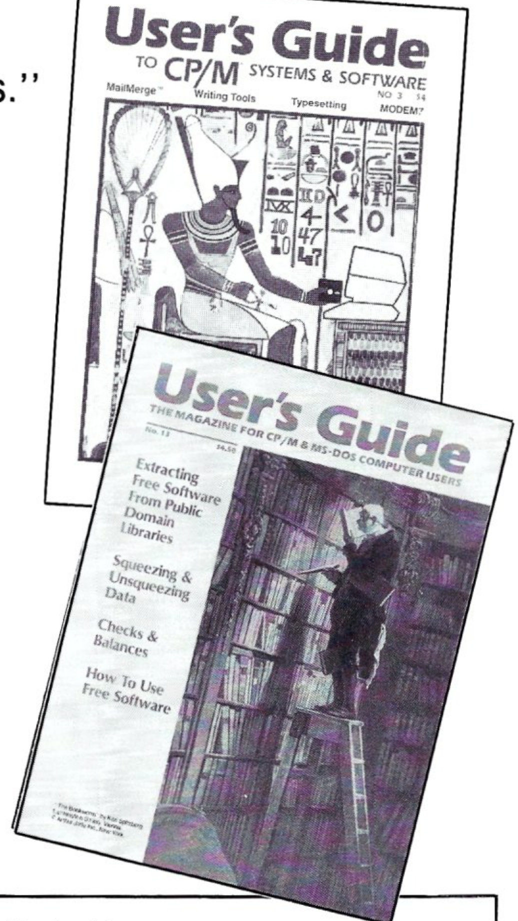
The following books, by Dave Cortesi, are among the best books on CP/M and contain many programming examples. They contain a great deal of info on installation, use of command sequences, file handling, directories, etc. And comprehensive reference sections are also included.

Inside CP/M"	$27.50 (US. Can. Mex) $37.50 (other foreign)
Inside CP/M-Plus	$19.95 (US. Can. Mex) $29.95 (other foreign)
Inside CP/M-86"	$18.95 (US. Can. Mex) $28.95 (other foreign)
Inside Concurrent CP/M	$18.95 (US. Can. Mex) $28.95 (other foreign)

Send check for full amount (NJ residents include sales tax). Shipping is included in price. Professional books may be tax deductible. All checks must be payable in U.S. funds by a U.S. bank. If billed to a firm or institution add $5.00 and include a Purchase Order Number. All books will be shipped within 2 weeks of receipt of order.

Mail to:
Micro/System Journal
Box 1192
Mountainside NJ 07092

We Explain Software Packages, Then We Teach You How To Use Them. ***User's Guide*** is the Magazine of Tutorials.



"...worth more than it costs."
Jerry Pournelle, ***BYTE***

User's Guide® magazine helps you use CP/M® and MS-DOS® application software on your personal or multi-user computer. For the low cost of a magazine subscription ($21), you get six issues stocked with tutorials and software evaluations ($4.50 each on the newsstand). No fluff, just direct, readable "how to use" information for users of computers that run CP/M software.

If you have a CP/M or MS-DOS computer, you need to know where to find good software and accessories. ***User's Guide*** has in-depth evaluations of commercial ***and free public domain*** software, so that you can choose the best buy.

Don't spend hundreds on training packages. ***User's Guide*** teaches you how to use the most popular programs on the market, such as WordStar®, SuperCalc®, dBASE II® and Modem7. You learn at your own pace.

Editors Tony Bove and Cheryl Rhodes are the acclaimed writers of several computer books on CP/M and WordStar. Contributing editors and columnists include other great writers who use computers extensively, such as Arthur Naiman, Steve Rosenthal, Jonathan Sachs and Kelly Smith. The writing is crisp, intelligent and informative, without an overuse of jargon.

User's Guide also keeps up with the changing personal computer market, with informative articles about communicating with other systems and transferring programs and data.

NO RISK OFFER:

Try ***User's Guide*** today. Fill in your name and address on the card next to this ad. Check the box marked "bill me" and drop in the mail; if you don't like the first issue, write "CANCEL" on your bill and keep it (or better yet — pass it on to someone who can use it). If you think you'll want it, save $3 and send in your payment with the card or coupon.

Try ***User's Guide*** as a training tool. Make your computer investment profitable.

User's Guide

P.O. Box 5245

Redwood City, CA 94063

Indicate Offer and Length of your ***User's Guide*** subscription:

Take Advantage of the Low Price for Enclosing Payment Now!
☐ I enclose **$21** for 6 issues
☐ I enclose **$40** for 12 issues

Or, try our no-risk bill-me offer and cancel if not satisfied!
☐ Bill me **$25** for 6 issues
☐ Bill me **$45** for 12 issues

☐ **Foreign:** (prepaid US $$): Canada & Mexico $27/six issues, overseas $40/six issues (airmail).

Payment by charge card: ☐ Visa ☐ Mastercard Exp. date ______________

Card # ______________ Interbank # ______________

Signature required: ______________

Name ______________

Company ______________ Title ______________

Address ______________

City, State, Zip ______________

☐ Check if you do not want promotional mailings.

Make checks payable to ***User's Guide***. Offer expires 12/31/85
Subscription Dept.: P.O. Box 5245, Redwood City, CA 94063.
Please allow 6-8 weeks for processing.

UG 13ad

• ***User's Guide*** is published by TUG Inc., with no strings attached to manufacturers or software distributors.

Building an IBM-PC or XT Clone

by Hank Kee

In the early days of personal computing it was quite fashionable to build your own system from various component parts. The reasons were normally for economy, experience, and the lack of availability of a packaged system. There was of course a certain pride of accomplishment in those days to see an "A>" prompt come up on the screen.

But time waits for no man. In a very short period of time, the hacker-dominated personal computing field gave way to the appliance-oriented personal computer market. It is however still possible to build your own system today. Building an IBM-PC or XT clone has become very easy. It is not even necessary to use a soldering iron. One can literally assemble an IBM-PC or XT clone system with only a screwdriver, nutdriver, and a reasonable measure of self-confidence.

With this in mind, some time ago, I decided to put together a system that would "clone" the IBM-PC. At the time I did it, IBM had not gone through the various iterations of price reductions. However, as clone component parts become readily available, IBM-PC system prices have also dropped drastically. If one were to stick with "equivalent" products, the differential, today, would be about $100. If, however, we were to substitute "lower" quality products, the savings could increase to about $300-$400. Is it worth the effort? And, how does one go about putting an IBM-PC clone system together?

The Motherboard & ROMS

One can buy bare, etched and drilled glass-epoxy PC-clone motherboards, devoid of components, with assembly instructions in the $80-100 range. Partly assembled boards, which include wave-soldered IC sockets generally run about $200. I think the additional cost is worth it. It eliminates the most time consuming and tedious part of the assembly work. Further, it significantly reduces the likelihood of problems that result from solder bridges or cold solder joints. It should also be pointed out that the PC uses a DMA circuit that is component sensitive and hence someone building up a system from bare board and has trouble getting the system to operate or operate reliabily should pay close attention to the operation of this circuitry. In some cases it has been found that some motherboards, because of layout or poor quality control are the culprit.

When contemplating the buying of a blank, partially assembled, or even a completely assembled motherboard it is a good idea to first check over the documentation that accompanies the board. Some, particularly from the far east, contain no assembly information or very poor documentation. Generally, if the documentation is good the product is good. If the documentation is poor, watch out!

One other consideration here is that many of the components that you will need are not readily available at flea markets or from many electronic component distributors. It may take quite a bit of searching, expense and time, to get all the parts together.

There are many PC and XT-like assembled and tested system motherboards selling starting at about $400 populated with 128K bytes of memory (a listing of many U.S. manufacturers or distributors of such boards will be found at the end of this article). They typically include all of the support chips. Many include features not found on the IBM-PC motherboard. For example, may accommodate more plug-in cards, may have sockets for more than 64K of ROM, may be able to take up to 1 Megabyte of RAM on board, may have a reset switch circuit, may have a kludge area, or may include the ports and display and disk controller circuits. Some manufacturers supply motherboards with an 80286 instead of the 8088 for improved performance.

Some motherboards however contain an empty socket for the BIOS ROM, which the purchaser has to supply. Most of the U.S. manufacturers of motherboards include a BIOS ROM. None include IBM cassette BASIC ROM's. However, sockets are included for additional ROMs. An additional 128KB memory is needed to bring it up to 256KB. PROM space is available to give us the same functionality of the IBM ROM monitor and ROM Cassette BASIC. IBM compatible BIOS ROMs are available from some suppliers starting at $30.

But the BASIC ROM's can present problems if it is an absolute requirement to build a 100% compatible clone unit. After all, we do not want to infringe on copyrighted materials*. Like all IBM-PC clones marketed today, the BASIC ROM's will have to be replaced with some version of Microsoft's GW-BASIC which is normally supplied on all copies of the MS-DOS system diskette. GW-BASIC runs completely off disk. If one buys a copy of IBM's PC-DOS one gets copies of BASIC and BASICA which use the IBM's ROM Cassette BASIC. These versions of BASIC will not run on the clones. It is worth buying a copy of IBM's PC-DOS just to get copies of their execellent manuals. However, keep in mind that you will also have to obtain a copy of Microsoft's GW-BASIC to avoid copyright infringement. This would have to be obtained independently from a software distributor.

You can buy a BIOS ROM from IBM. IBM sells the part to users who wish to upgrade their systems from the BIOS used in the old PC revision A (64KB) motherboards to the newer BIOS. These ROM's however have 24 pins and will not work with the 28 pin PROM circuitry used on clone system

motherboards. Further, the ROM IBM sells is for the PC and not the XT. The clone boards seem to be mostly XT-like. Thus 100% compatibility is not achieved.

The Box & Power Supply Considerations

There are many suppliers of boxes to house your motherboard, power supply and drives. Outwardly they look just like the IBM-PC box. However, some have subtle differences that can cause a great deal of work to make them accommodate your motherboard. Further, many of the clone motherboards are not the same dimensions as the IBM-PC board and hence may not fit a box that conforms to the IBM-PC dimensions. Be forewarned, check the dimensions of your motherboard, power supply and cabinet to be sure that they will all fit together easily.

An XT-like power supply can be purchased anywhere from $145 to $175. But be wary of these units. Some of them are not equipped with the power socket to accommodate the monochrome display.

The XT power supply is rated 130 watts. The clone units may also claim 130 watts. But it is most important that the individual 5 volt and 12 volt lines have sufficient power. Therefore check their ratings.

Some power supplies come only with two disk drive power connectors. Thus, one must obtain a Y-connector cable if two floppy drives plus a hard disk drive are to be used. However some of the power supplies come with 3, and some even with 4 power connectors and cables. This is very convenient if you are planning to have 3 or 4 drives in your systems. It eliminates an extra Y connector(s) for the extra drives.

The Box

System chassis cases range in price from $70 to $110. Do not expect an IBM logo on the front of these boxes. They are quite adequate but usually they lack the equivalent fit and finish of the IBM enclosure. Further, most of these clone boxes use metric threading. This may or may may cause a problem with the installation of some components. Also, you may have difficulty find metric threaded nuts and bolts, if not enough hardware is provided.

Keyboards

The next item to consider is the keyboard. Some of the clone keyboards are available for under $100. Although the original IBM keyboard lists for $275, they are available for $200 in large metropolitan areas. My preference is the IBM keyboard for the tactile touch. However for a difference of $100, you can sure adapt to another keyboard.

There are other keyboards priced in $100-200 range which includes LED's for CAPS LOCK, and NUM LOCK. And, on some keyboards the awkward backslash (\) has also been relocated to a more convenient location.

Try not to buy in the blind. Test type and feel these keyboards prior to purchase, if at all possible.

Drives

Now add to it diskette drives. Half-height drives are available for as low as $100 each, but $125 is more typical. I have found them to be reliable. My only objection is that they require 3 millimeter screw fittings that are not supplied. When I contacted an importer about this problem they told me to go to an import car supplier. My drives presently sit loose. It's not the way to mount drives, but neither can I buy the necessary screws in quantities less than 100.

Full height diskette drives are available for about $150 if you do a little shopping around. Also, figure on about $25 for the diskette cable to connect the units. An IBM diskette controller board lists for $125. Do not figure on saving much money here by buying a clone diskette controller card.

Pricing the Total System

So what does the total expense look like compared to a similar IBM PC system? An IBM PC with the same options sans display and adapter sells for about $1495 to $1595.

Here is a price breakdown.

	rock-bottom	equivalent	IBM
systems board	$400	$400	
additional memory	70	70	
BIOS ROM	30	30	
chassis	70	110	
power supply	145	175	
keyboard	100	200	
half height drives	240	300	
diskette controller	100	125	
cabling	25	25	
	1180	1435	1495-1595
savings	315-415	60-160	
compatibility	(less than 100%)		real McCoy
connector slots	8	8	5

Worth the Savings?

Is the effort worth it? As an academic exercise, the answer is no. The only time to consider building your own clone is when more than 5 connector slots will be required and all the software you need can run with the provided BIOS ROM. How will one know if the desired software will run? The only way to find out is to find someone who may have put together a clone with that particular BIOS ROM in question.

At one time, IBM prices were sufficiently high that many IBM clones came to market. Not only has price reductions affected IBM clones causing many to get out of the business, but it has also impacted the put-it-together market if one were to carefully analyze the net expense. For me the building of an IBM clone was an experience. If I had known of the drastic price reductions that were to occur, I would have saved quite a few dollars by waiting.

One last point. If you are building a custom system (e.g. for a process control application) putting together a PC-clone makes real sense. You can develop your software on a standard PC system, burn the program into ROM(s) and then plug them into a clone board for the actual application. In fact, the actual application may not even need a display, keyboard or drives. This makes for a much lower cost custom system then buying a PC, which is probably overkill for the the task.

U.S. Manufacturers of PC/XT Compatible Motherboards

Advanced Computer Solutions Inc.
13720 Midway Rd Suite 209
Dallas TX 75244
(214)934-8239

Display Telecommunications Corp.
4100 Spring Valley Rd Suite 400
Dallas TX 75234

Faraday Electronics
743 Pastoria Ave
Sunnyvale CA 94086
(408)749-1900

Handwell Corp.
4962 El Camino Real
Los Altos CA 94022
(800)821-3628

Oemtek
3707 Williams Rd
San Jose, CA 95117
(408)247-1100

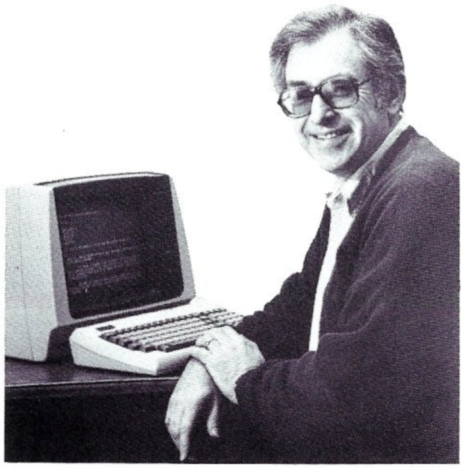
Slicer Computers Inc.
2543 Marshall St. N.E.
Minneapolis, MN 55418
(612)788-9481

Super Computer
17813 S. Main St. Suite 103
Gardena CA 90248
(213)532-2133

Wave Mate Inc.
14009 S. Crenshaw Blvd.
Hawthorne CA 90250
(213)978-8600

**Editor's Note: An IBM-PC compatible ROM BIOS was developed*

(continued on page 80)



For the Advanced Computer User
Micro/Systems Journal™
P.O. Box 1192
Mountainside, N.J. 07092
(201) 522-9347

Sol Libes says...
"Call me crazy
...I'm doing it again!"

I'm back into magazine publishing ... something I swore I would never do again. When Microsystems magazine died I was in a depressed state for weeks. Countless letters and phone calls from devoted subscribers made it even worse. Everyone kept urging me to do it again.

I kept remembering what my wife Lennie and I went through when we started Microsystems, and I said no ... not again ... we want to live a normal life again. But there has been something missing from my life the last several months. The passing of Microsystems left a void. Let's face it, there really is no other magazine that caters to the advanced micro user the way Microsystems did.

So I am starting a new magazine. It is in the tradition of the old Microsystems. Lots of practical info ... strictly technical ... no fluff ... stuff to keep every hacker up-to-date on the ever-changing micro technology ... software and hardware tutorials and reviews, public-domain software info and reviews (SIG/M, PC/Blue, PC-SIG, C-User Group and more) ... MS/DOS, CP/M, Turbo DOS, C, Pascal, Forth, Lisp, and of course Assembler ... S-100, IBM-PC, single board computers, multi-user systems ... a real micro systems-oriented journal ... in fact, that is its name — ***Micro/Systems Journal!***

Micro/Systems Journal *is published six (6) times a year and I am offering a special introductory rate. Subscribe before April 15th and deduct 10%. This offer is good until April 15th so hurry and mail your subscription now.*

For the Advanced Computer User
Micro/Systems Journal™

Subscription Form

☐ **Yes! Start my subscription!**

	Bulk Rate	First Class
☐ 1 Year	$18.00	$24.00
☐ 2 Years	32.00	44.00
☐ 1 Year (Canada & Mexico)		24.00
☐ 2 Years (Canada & Mexico)		44.00
☐ 1 Year (other foreign)		32.00
☐ 2 Years (other foreign)		58.00

☐ Postmarked before April 15th .... **deduct 10%**

☐ Former Microsystems subscriber .. **deduct an additional 5%**

Name: ____________________

Address: ____________________

____

City: __________ State: __________ Zip: __________

Please make check payable in U.S. funds by a U.S. bank. Thank you.

☐ Purchase Order #__________.
Bill my firm and add $5 billing charge.

Make check payable to: Micro/Systems Journal, P.O. Box 1192, Mountainside, NJ 07092

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.

NOVEMBER: Intro to 80286, 68000, and 16032 Microprocessors; Intro to Local Area Networks - Part 2; Extended Memory Management for older S-100 Systems; Notes on Microsoft Fortran-80; Building S-100 Parallel Ports; REVIEWS: CompuPro CPU-68K, System 8/16, Xenith Z-100, Nevada & Ellis Computing Fortran.

OCTOBER: Intro to Local Area Networks, Part-1; Build Low-Cost LAN; Build S-100 Bubble Memory Card; Use Radio Shack Model 100 portable with a CP/M system; Write Menu-Driven Utility for Setting Printer Options; North Star Improvement; True Z-80 Random Number Function; Hide Code in Basic REM statements; Machine Code loader for MBasic; Increase Single-Density Disk Formatting; Relocating Assembler & Linkage Editors, Part-2; Run MX-80 with North Star; User Group Directory; CP/M-86 Versus CP/M-80; REVIEWS: CP/NET, QBAX, S-Basic.

SEPTEMBER: Using RatFor; Relocating Assemblers & Linkage Editors, Part-1; Sleuth WordStar Files with Pascal; CrossCheck Program; CP/M<->NorthStar File Transfers; NorthStar DOS as a CP/M COM File; Add Rescue Key to System; S-100 TMA Interfacing; REVIEWS: Altos 586, CompuPro 8/16C, Ithaca InterSystems Encore, Dual Systems 83/20 Unix System, Supersoft C, Software Tools, Morrow Designs Micronix, Upgrade Older S-100 Systems to CompuPro Dual Processor.

AUGUST: XERA Program; Logging-On CP/M; WordStar Date/Time Patch; Find Location of Variable in NorthStar Basic; Prevent System Crashes During Warm Boot; Enhance Spreadsheet Print Files; Plotting Package-Part 3; Run WordStar under TP/M; 50-line Text Formatter; Using the LU Utility; User Areas under CP/M; REVIEWS: Stiff Upper Lisp, MuLisp-80, Supersoft Lisp, Cromenco C-10, Access Manager, Fancy Font, Computime SBC-880 S-100 card.

JULY: Using RCPMs; RCPM Directory; PIP Data Between Computers; Toward Smarter Modem Programs; Interface MX-80 via Parallel Interface; Digital Audio On CP/M System; Customize CP/M CBIOS; Plotting Package Part-2; REVIEWS: DRI PL/I-86 and PL/I-80, S-100 PMMI MM-VT1.

JUNE: Plotting Package Part 1; Drive HP Plotter; Laboratory Graphics Applications; Console Keypressed interrupts; Cutomize Wordprocessor Keyboard; WordStar Patch hor H-19/Z-19 Terminal; Relocatable Code; REVIEWS: Graftalk, JES S-100 Graphics Controller, ZCPR2.

MAY: IEEE-696/S-100 Standard Update; S-100 Product Directory; Solid State Disk Drives; Track-Buffering for Tarbell SD Controller; SID Patches; Using Microsoft's VARPTR; Double NorthStar RAM; Restore unsaved MBasic Programs; Pascal Disk Scan Program; Extended Memory for Exidy Sorcerer; S-100 Extended Addressing; Unix Easy Applications and Text Processing; REVIEWS: NorthStar Basic Utilities, CompuPro MPS-1, Autodiff, S.A.I.L., IBIOS.

APRIL: IEEE-488 Tutorial; Interfacing to Lab Instruments; CP/M-86 System in Lab; Implementing CP/M+ Part II; Build Simple S-100 Card Extractor; Macros & MacroAssemblers; REVIEWS: Pickles & Trout S-100 488 Controller; CP/M Utilites; Morrow Decision I.

MARCH: Implementing CP/M+ Part 1; Two CP/M Enhancements; Transferring Files between 5'' and 8'' drives; NorthStar DIRALPHA program; Large BIOS Problem; WordStar Mod For Epson Printer; SpellStar Bug; Better Multiply Algorithm; REVIEWS: WordStar, Wordmaster, Magic Wand, Spellbinder; Televideo 925, Zenith Z19, Wyse WY-100 and ADDS Terminals.

FEBRUARY: CP/M+ Overview; Implmenting CP/M+; Triple Floppy Density; CP/M Chain Routine; Troubleshooting S-100 Systems; Build S-100 EEProm

Microsystems Back Issues *continued*

Card; Relocatable Code; REVIEWS: Ackerman S-100 Promblaster; Pascal MT+; Four S-100 Single Board CPUs; Morrow Micro Decision.

JANUARY: Unix Vs CP/M; Intro to Xenix; Unix on Micros; Build S-100 DMA Adaptor; Interfacing to BSR X-10 Home Control System; S-100 Troublehooting; REVIEWS: InterSystems DPS-8000 and Coherent; MicroShell, UNICA, Small-VOS, Small-Tools, Five S-100 RAM Cards, SemiDisk.

1982

NOVEMBER/DECEMBER: CP/M Vs MS/DOS; CP/M-86 Vs MS-DOS; Intro to ADA Part 2; Virtual Disk for NorthStar; CP/M Program Auto-execute; Macros & Macro-Assemblers; REVIEWS: Janus, Aztec-C, C/80, Morrow S-100 M26 Hard Disk System, Teleram S-100 Bubble Memory Card, Jade S-100 Bus Probe.

SEPTEMBER/OCTOBER: Innovations in Micro Languages; Intro to Ada Part 1; Saving Program State Under UCSD Pascal; Intro to Stoic; Stoic Vs Forth; Intro to C; Build S-100 Timer Interrupt for MP/M; REVIEWS: UCSD Pascal VII & IV.

JULY/AUGUST: Hardware Random Byte Generator; Error Detection & Correction Codes; Getfile CP/M Utility Program; CP/M Patches; CP/M Application Notes; Run old NorthStar programs under new DOS; Cloning Disk Drives; Low Cost Floppy Disk Power Supply; Intro to Computer Graphics; Using Supersub Utility; REVIEWS: D80, RAID-8080, Three Macro-Assemblers, PDS, Cer-Tek S-100 UniProm Board; GrafPak.

MAY/JUNE: Intro to DBMS; Three ways to implement a mail list; Cursor Addressing; Structured Programming in Basic; Replacement for CP/M Submit; CP/M Disk Directory & Table Secrets; Mods for SDS VDB-8024; Run NorthStar Basic with CP/M; REVIEWS: DataStar, MDBS, TIM, Mince, ZDM.

JANUARY/FEBRUARY: Intro to PL/I-80; Programming Styles; Interfacing PL/I-80 to Assembly Language; Intro To C Part 2; 6-byte Hex-ASCII Conversion; Little ADA Part 3; Use Computer To Build Computer; 65K RAM for Sol-20; Using CP/M's Autoload Feature; REVIEWS: PL/I-80, Diskindx.

1981

JULY/AUGUST: 16-Bit Disk Operating Systems; Input Queuing For NorthStar; Variable Speed Automatic Slow Step; Build S-100 Clock/Calendar Card; REVIEWS: TEC-86 System, Seattle Computer 8086 System, AlphaMicro, Godbout Dual Processor, CP/M-86, Televideo 920-C Terminal.

MAY/JUNE: Intro To Computer Communications, CP/M Tutorial Part-5; CP/M Enhancements; REVIEWS: Modkom, Commx, Mcall, S-100 Modems Compared.

1980

MAR/APRIL: Linear Programming Techniques in Pascal; Intro To CP/M part 2, Addressing The Cursor; S-100 Bus - New Vs Old; Tarbell Disk Controller Mods; REVIEWS: CGS-808 S-100 Color Graphics Controller.

CLASSIFIEDS

Micro/Systems Journal will accept Classified Ads. The charge is $6/line (3 lines minimum); 50 characters maximum per line. Three times Frequency $5/line; six times $4/line; non-profit clubs $2/line. Logos, special type, etc., are an extra charge. Check must accompany ad copy. Send to **Micro/Systems Journal**, POB 1192, Mountainside NJ 07092.

PC CLONE
(continued from page 76)

some years ago by the Taiwan "Institute For Information Industry" for use by Taiwan manufacturers of IBM-PC compatible machines. This BIOS was licensed to these manufacturers and several have made units containing these ROMS. However, the U.S. Customs has, reportedly, confiscated systems (at IBMs request) that arrived in the U.S. Further, IBM has filed suit against 11 Taiwan companies for copyright infringement. This case is currently in the Taiwan courts and is, reportedly being fought by the companies.

There are an estimated 50 Taiwan system and component suppliers of PCs and PC-compatible components. Many are already exporting units to the U.S. In most cases these units lack BIOS ROMs. However, the likelihood is that this situation will change shortly.

Most of the board manufacturers listed in this article include a ROM BIOS with their motherboards.

Hank Kee is a Vice President for a very large banking institution. He is also the software librarian for the PC/Blue public domain software library and served as the librarian for SIG/M, in its early days.

Advertiser Index

They Called It Impossible.

We Call It TurboLAN.™

Imagine IBM PCs, intelligent workstations and S-100 BUS multiuser systems all on the same local area network.

Impossible?

Not with TurboLAN, Intercontinental Micro's networking solution.

The key is a unique hardware and software approach to local area networks.

You get the flexibility of 8-bit and 16-bit processors on the same network, the cost savings of shared peripherals, and expandability to over 4000 users.

You simply organize the network the way you want, mixing IBM PCs, S-100 BUS multi-user systems, S-100 BUS file servers, and workstations.

Intercontinental's sophisticated single board computers, 8-bit and 16-bit slaves, memory and interface boards make your job even easier. TurboLAN will work with almost any S-100 BUS system including NORTHSTAR, IMS, Teletek, and any other S-100 master featuring phantom deselection or extended addressing capability.

The ability to run PC-DOS, MS-DOS, CP/M, MP/M, and other applications programs means whatever the future holds, TurboLAN is ready.

And so is Intercontinental Micro.

You see, our S-100 BUS products revolutionized an industry. Intercontinental's slaves, masters and memory boards introduced advanced features like Direct Memory Access and Memory Management to micro environments. TurboLAN.™ The new networking solution from the company that's been building networking solutions for years.

Still think TurboLAN's impossible?

Then call, write or circle the bingo number below — we'll send you complete information on TurboLAN and descriptions of all our products, including the most complete S-100 BUS product line in the business.

OUR COMPLETE TURBOLAN AND S-100 BUS PRODUCT LINE

CPS4800X-SINGLE BOARD COMPUTER.
4 or 6 MHz processor (Z80A/B), onboard floppy disk controller, 64K RAM, 4 channel DMA controller, 24 line parallel I/O port, two serial I/O channels, real time clock. Memory mapped or I/O mapped capability.

CPS-16X — 16-BIT 8086 SLAVE PROCESSOR.
256K, 512K or 1MB RAM, bank selectable, memory mapped, two serial ports, 24 line parallel I/O port, 8 MHz.

CPS-BMX/MX — Z80 SLAVE PROCESSORS
4 to 6 MHz processors, 64K RAM or 128K bank selectable memory, two serial ports, 24 line parallel I/O port.

256KMB — MEMORY BOARD.
Hard disk cache, linear addressable to two megabytes, bank selectable in 16K increments, configures for phantom deselection, parity error detection.

LANS-100 — TURBOLAN FOR S-100 BUS SYSTEMS
ARCNET controller meets 696.2/D2S-100 spec, coax cable interface, 255 nodes per network segment, 2.5 megabit/sec. data rate.

LANPC TURBOLAN FOR THE IBM PC
Plug-in expansion ArcNet board. Custom software drivers integrates IBM PC into TurboLAN networks. 64K or 256K RAM options available.

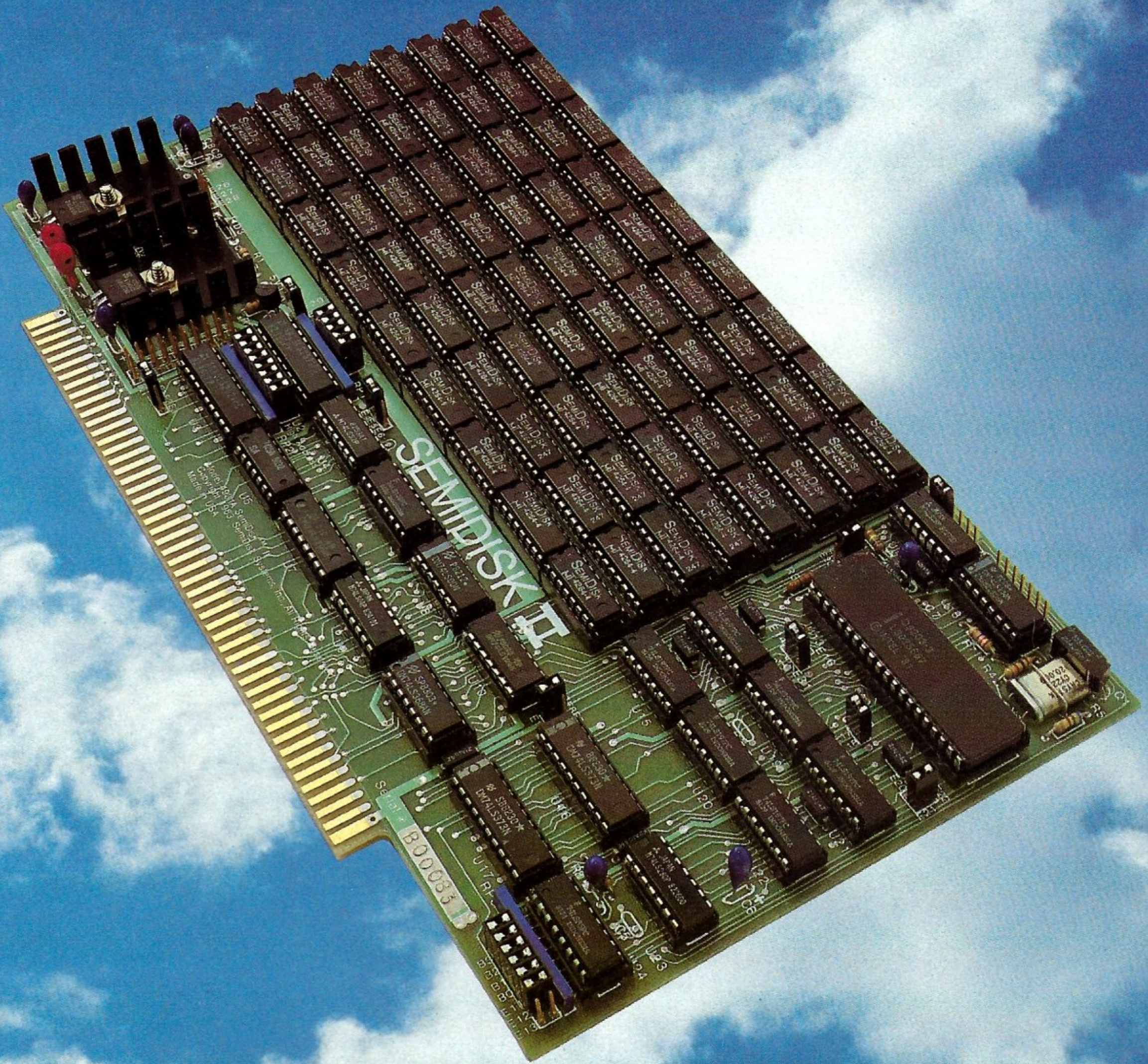
WS80X-DISKLESS WORKSTATION
Converts almost any dumb terminal into intelligent workstation with networking capability. Floppy and hard disk options available.

PERSONALITY BOARDS —
SASI, Centronix, PRIAM, Clock/Calendar, RS232, Modem, RS422, long distance serial communications (up to 4000 Ft.)

Intercontinental Micro Systems

4015 Leaverton Ct., Anaheim, CA 92807, (714) 630-0964, TELEX: 821375 SUPPORT UD

2 Megabytes



THE LARGEST CAPACITY DISK EMULATOR YOU HAVE EVER SEEN.

You know about disk emulators. They're fast semiconductor disk drives. Very fast. But until now, the most disk storage you could get on a single board was 1Mbyte. (That was from us, too.) Now we have news that'll really blow your socks off... 2 Megabytes on a single board. Available NOW. That's not a pie-in-the-sky promise.

That's enough storage for dozens of large programs and hundreds of kilobytes of data files. Enough for almost anything you want to do with a disk drive. But that's not all. With SemiSpool, our CP/M print spooler, you can implement a print buffer hundreds of kilobytes long in seconds. All in software. At no extra cost.

Another thing about disk emulators. Unless they're from SemiDisk Systems, they're probably afraid of the dark: Lose power or turn the computer off, and your valuable data goes to that big backup disk in the sky. But our Battery Backup Units keep SemiDisk data flying high while your computer is off, and up to 10 hours during a complete blackout.

So remember this: SemiDisk Systems has been building dedicated microcomputer disk emulators longer than anyone. And larger. And faster. And at a much lower cost. And that's not a lot of hot air.

AT A PRICE YOU NEVER THOUGHT YOU'D SEE

	512K	1Mbyte	2Mbyte
SemiDisk I, S-100	$995	$1795	
SemiDisk II, S-100	$1295	$2095	$2549
IBM PC, XT, AT	$945	$1795	$2499
QX-10,QX-16	$799		$2499
TRS-80 II,12,16	$995	$1795	$2499
Battery Backup Unit	$150		

SEMIDISK

SemiDisk Systems, Inc.
P.O. Box GG, Beaverton, Oregon 97075
503-642-3100

Call 503-646-5510 for CBBS/NW, 503-775-4838 for CBBS/PCS, and 503 649-8327 for CBBS/Aloha, all SemiDisk-equipped computer bulletin boards (300/1200 baud). SemiDisk, SemiSpool trademarks of SemiDisk Systems. CP/M trademark of Digital Research.