the CP/M* and S-100 user's journal

$2.50 U.S.A.

MICROSYSTEMS™

JAN/FEB 1982

VOL. 3/NO. 1

INTRODUCTION TO THE PL/I-80 PROGRAMMING LANGUAGE

See Pages 28-46

Also in this Issue

and more

Complete Table of Contents on Page 3

*CP/M is a registered trademark of Digital Research.

MORROW DESIGNS

Now, Complete 5¼" System
5 Mbytes $2,495

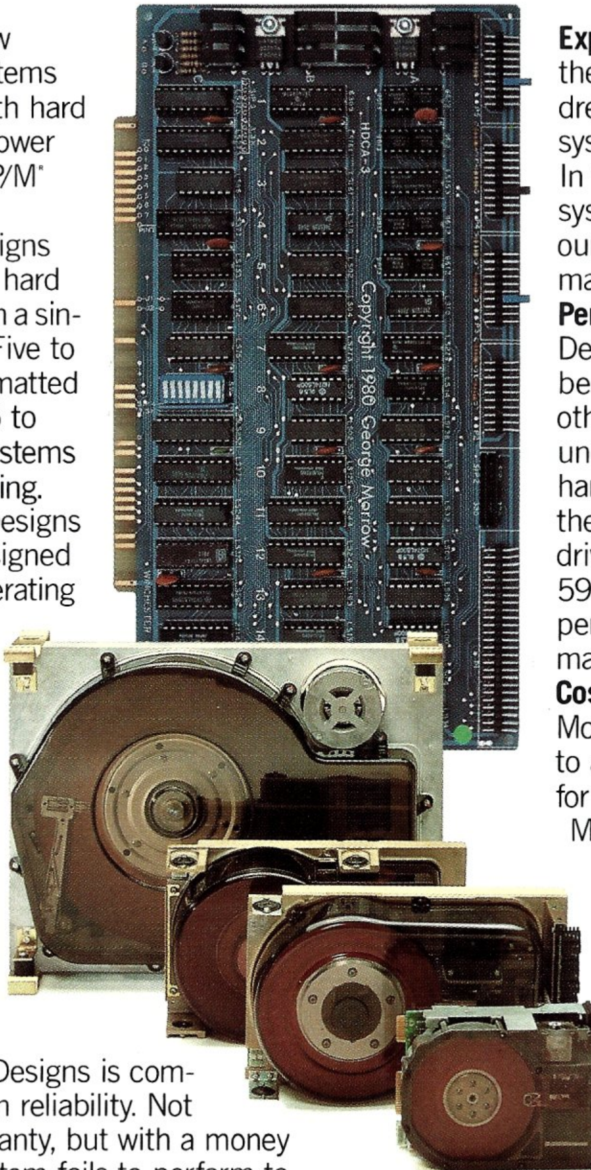
Leading edge technology in hard disk systems.

Complete systems. Morrow Designs hard disk subsystems are delivered complete with hard disk, controller, cabinet, power supply, fan, cables and CP/M* 2.2 operating system.

Widest range. Morrow Designs offers the widest range of hard disk systems available from a single supplier. 5¼," 8," 14." Five to over 100 megabytes of formatted hard disk storage. $2,995 to $17,980. Cost effective systems that work. And keep working.

S-100 and more. Morrow Designs hard disk systems are designed for use with the CP/M operating system. Available software packages allow our systems to run on any IEEE696/S-100 Standard system with no hardware modification. Plus, Cromemco,*** North Star,** Vector Graphics, Godbout, Dynabyte, Exidy,**** IMSAI, Micromation, Processor Technology and California Computer Systems.

Reliable systems. Morrow Designs is committed to hard disk system reliability. Not simply with a 90-day warranty, but with a money back guarantee. If our system fails to perform to specification, send it back. We'll send back your money.



Experience. As of April, 1981, there were over fifteen hundred Morrow Designs hard disk systems successfully installed. In fact, over 200 independent systems integrators now use our hard disks to solve their mass storage problems.

Performance answers. Morrow Designs hard disk systems have been benchmarked against all other systems. None is faster under CP/M. Morrow Designs hard disks operate at 10 times the speed of a floppy disk drive. Transfer rates range from 590,000 bytes to 900,000 bytes per second. That kind of performance can become addictive.

Cost effective answers. Compare Morrow prices and performance to anything presently available for S-100 systems. You'll find Morrow's price/megabyte/performance ratio to be unmatched. Leadership in disk systems technology earned us leadership in price/performance. And that may have earned us a call from you. Circle the Reader Service Number for our full line data sheets. Can't wait? Call us at (415) 524-2101. And yes, OEM quantity prices are available. **LOOK TO MORROW FOR ANSWERS.**

MORROW DESIGNS

5221 Central Avenue, Richmond, CA 94804
(415) 524-2101

*CP/M is a trademark of Digital Research.
**Northstar is a trademark of North Star Computers, Inc.
***Cromemco is a trademark of Cromemco, Inc.
****Exidy is a trademark of Exidy Corporation.

MicroDaSys

68K MiniFrame Power

S-100 Interface Now Available!

features:

- Virtual Memory Management
- Motorola 68000 Processor
- 12 Mhz operation
- 32 bit computations
- 4 Billion byte addressing
- LRU demand paging
- Peripheral processor
- UNIX multi-user operating system

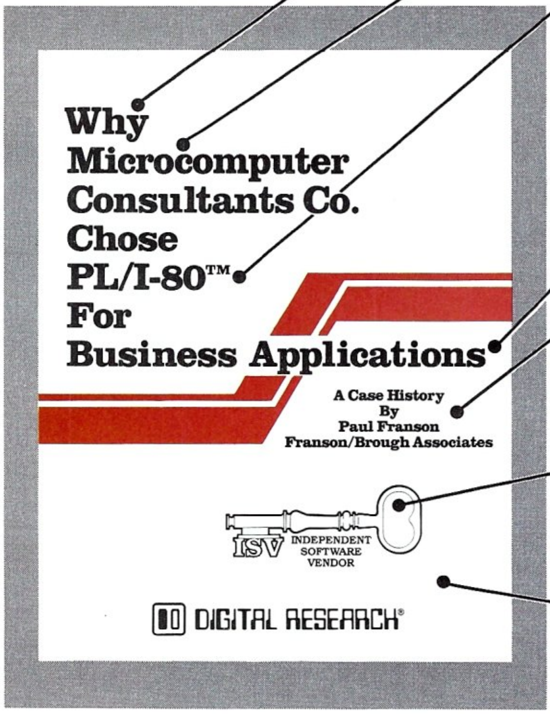
Only the world's finest minicomputer offers these features in systems available TODAY. Complete single-user systems start under $10,000. And a 6 user UNIX-based 16 MB virtual system with 512K Bytes RAM and 24M Bytes hard disc costs under $25,000! Want proof? Call or write now for brochures and information on our 68K Seminars. Or send $75 for the complete technical manual set.

MicroDaSys
2811 Wilshire Blvd., Santa Monica, CA 90403
Call (213) 829-6781

TWX: 910-321-2378

A free poster of this ad is available while supply lasts.
UNIX is a trademark of Bell Laboratories.
68K MiniFrame is a trademark of MicroDaSys, Inc.

PL/I-80™: Not for beginners*



Why Microcomputer Consultants Co. Chose PL/I-80™ For Business Applications

A Case History
By
Paul Franson
Franson/Brough Associates

DIGITAL RESEARCH®

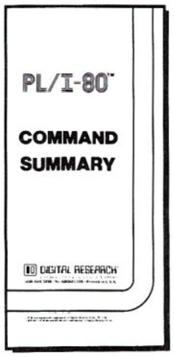
- **For internal savings,** product capabilities, code speed and upward compatibility — How PL/I-80 meets these criteria is making software news.
- **MC is a successful company** selling business applications to first time users.
- **MC switched** from a popular but less efficient language. MC benefits from these PL/I-80 (ANSI Subset G) features. • *Fast design* with picture specifications for output. • *Easy data retrieval* with BT-80™ file manager. • *Improved performance* with LINK-80™ loading of disk resident overlays. • *Small, fast programs* with optimized object code and no interpreter.
- **Specific examples** of PL/I-80's speed, memory conservation, error checking, and control in the business environment are cited.
- **In a candid interview,** Mr. Charles Russell, MC's president, discusses his professional programming needs.

* *Mr. Russell distinguishes, "PL/I-80 isn't a panacea. It's a programmer's language, ideal for the experienced programmer. Just as PASCAL and BASIC are good for beginners."*

- **Worldwide market exposure** of your products and company through your listing in our CP/M® Compatible Software Catalog; news affecting your business in a free subscription to ISV FORUM; plus comprehensive seminars comprise our ISV Support Plan for those who write applications under CP/M.

Yours free. See why MC switched. Then compare your language.

Limited quantity. Respond now and receive a bonus book, *PL/I-80 Command Summary,* a quick reference to 343 PL/I-80 commands and error messages.



PL/I-80™

COMMAND SUMMARY

DIGITAL RESEARCH

I'd like to analyze PL/I-80 features. Send your Brochure & Bonus Book to:

Name:____________________

Title:____________________

Company:____________________

Address:____________________

____

DIGITAL RESEARCH®
P.O. Box 579V, Pacific Grove, CA 93950

Europe: Vector, Intl., Leuven, Belgium, 32(16)202496
Far East: Microsoftware Assoc., Tokyo, Japan, 03-403-2120

CP/M is a registered trademark of Digital Research, Inc. PL/I-80, BT-80 and LINK-80 are trademarks of Digital Research, Inc.

the CP/M* and S-100 user's journal

MICROSYSTEMS™

Volume 3, Number 1 January/February 1982

Staff

Sol Libes....editor
Claudette Moore....managing editor
Chris Terry
Jon Bondy
Anthony Skjellum....contributing editors
Don Libes
Fred Gohlke
David Gewirtz
Randy Reitz....assisting editors

David Ahl....publisher

Laura MacKenzie....production manager
Sue Gendzwil....art director
Diana Negri....assistant art director
Chris DeMilia
Carol Ann Henderson
Eugene Bicknell....layout
Jean Ann Vokoun
Maureen Welsh....typesetting

William L. Baumann....operations manager
Patricia Kennelly....personnel and finance
Ethel Fisher....bookkeeping
Elizabeth Magin....secretary
Frances Miskovich
Carol Vita....circulation
Sue DeMark....customer service
Jim Zecchin....order processing
Jennifer Burr
Laura Gibbons....retail sales
Mary McNeice....office assistant
Ron Antonaccio....shipping & receiving

Editorial correspondence is welcomed and should be sent to: Sol Libes, c/o MICROSYSTEMS, Box 1192, Mountainside, NJ 07092. Phone: (201) 522-9347.

For information on commercial advertising, write to: MICROSYSTEMS, 39 East Hanover Ave., Morris Plains, NJ 07950, or call Claudette Moore at (201) 267-4558.

MICROSYSTEMS (ISSN #0199-7955) is published bimonthly by Creative Computing, 39 East Hanover Ave., Morris Plains, NJ 07950. Second class postage paid at Morris Plains, NJ 07950 and additional mailing offices.

POSTMASTER: Send address changes to: MICROSYSTEMS, P.O. Box 789-M Morristown, NJ 07960.

Subscriptions are $16 a year (6 issues USA). To subscribe, call our toll free number: (800) 631-8112, or (201) 540-0445 in New Jersey.

In This Issue

The PL/I-80 Programming Language

Departments

NEWS & VIEWS

S-100 Standard Nears Adoption

The IEEE-696/S-100 Bus Interface Standard has been finalized by the working committee. The committee is now in the process of putting together the formal presentation to the IEEE Computer Standards committee, for adoption of the standard. When the standard is adopted, it will be published in *Micro* magazine, the offical IEEE microcomputer publication. It is anticipated that the IEEE will again give us permission to reprint the standard after its appearance in *Micro*, as they did when we reprinted the original proposed standard two years ago. Regretfully, the IEEE has prohibited us from publishing the standard until then. In the meantime, if you have a copy of the original proposed standard (which appeared in the July 1979 issue of *Computer* and January 1980 issue of *Microsystems*) and just want a copy of the addendum (14 pages) just send $2 ($3 foreign) to Sol Libes, Box 1192, Mountainside, NJ 07092.

Oasis News

Phase One Systems, suppliers of the Oasis DOS, are now publishing a quarterly, sixteen page newsletter called "Software Monitor," For a copy write: Phase One Systems, 7700 Edgewater Drive, Oakland, CA 94621; (415)562-8085.

The Oasis Users Group has released seven volumes of software which include utilities and applications programs. An Oasis User Meeting was held in November in Las Vegas. Membership in Oasis-UG is $35/yr, and includes the first software library volume (additional volumes are $35 each with credit given for software contributions) and a newsletter. Contact Oasis-UG, c/o Fred Bellomy, Box 2400, Santa Barbara, CA 93120; (805)965-0265.

Pascal/MT User Group Formed

The Pascal/MT Users Group (MTPUG) was recently formed. A quarterly newsletter will be distributed containing bug reports, fixes, programs and new items. Long Pascal/MT programs will be available on disk, 8" SD, and 5-1/4" NorthStar and Zenith/Heath formats. Dues are $7/yr U.S., $8/yr Canada and Mexico, and $10/yr other for surface mail, $16 air mail. MTPUG, Box 192, Westmont, IL 60559. In Europe, write MTPUG; Schimmelmannstr, 37A; D-2070 Ahrensburg, West Germany; dues are 25 DM.

SIG/M & CPMUG News

Last month the SIG/M User Group issued six more new disks, bringing their total up to 48 volumes. The CPMUG is issuing 20 new volumes. However, I have been informed that these volumes (55 through 74) contain the software from the first 20 volumes of the SIG/M-UG. The SIG/M-UG has given CPMUG permission to distribute their software. This will give the SIG/M-UG public domain software greater world-wide distribution and relieve the SIG/M-UG of some of its workload. However, CP/M users who deal with both organizations should take care not to order the same disks from both organizations. Hopefully, CPMUG will clearly label which disks are duplicates. For information on the SIG/M and CPMUG refer to "The Editor's Page" in this issue.

Lifeboat Switches

There is no doubt that Lifeboat Associates is the largest world-wide distributor of CP/M and CP/M-based software. To a great degree, Lifeboat can boast that they made CP/M the "software bus" by implementing CP/M on about forty different 8-bit microcomputer systems.

But when it comes to 16-bit microcomputers, Lifeboat has decided against CP/M-86. Rather, it has decided to support the Microsoft DOS being used on the new IBM personal computer. The rumor is that IBM selected the MS-DOS over CPM-86 because the financial terms were better. Apparently Lifeboat feels that with IBM support, this DOS will be much more popular than CP/M-86. Lifeboat will call their version of the DOS "SB-86," and will make it available for a wide variety of systems. Lifeboat will also convert most of their current CP/M-based packages to run under SB-86.

Rumors! Rumors!

SuperSoft Associates, of Champaign IL is rumored about to release a Z-80 ADA compiler. Rumors are that it will be a subset of ADA (actually the Department of Defense has still not finalized the complete ADA standard) and that

ANNOUNCING A REVOLUTION IN THE COST OF PROFESSIONAL SOFTWARE

VISACCOUNT is a fully integrated business and accounting system designed for use in small businesses. VISACCOUNT is extremely comprehensive and professional, yet it is very easy to use. The system is controlled from a series of interconnected menus permitting user-friendly operation. Everything you need to set-up and operate the system is provided with the VISACCOUNT package. Experts have estimated the development costs for a fully integrated software system ranges between $7,200 and $22,000.† When you buy software the developer has to recapture this expense. Computer Services Corporation of America is selling its software with a view that volume sales can almost negate this development cost.

OUR GUARANTEE — Buy both our software and that of our competitors (who will no doubt charge several times our price because they need to recapture their development cost). Compare the two systems and we know you'll return theirs (make sure they'll let you return their software). If you decide not to keep our system, then return it within 45 days for a full refund. Once you've used our system we're confident you'll be delighted.

†Microcomputers for Business, Applications, 1979.

VISACCOUNT

What You Receive

- Nine 5¼" double density disks (or six 8" single density disks)
- Easy-to-use operator's manual (over 200 pages)
- Self-study guide on bookkeeping and accounting (over 180 pages)
- Cassette based instruction program on set-up and operation

Available for Apple*, TRS-80, and most others

*The Apple version requires the Microsoft Z80 softcard.
CSCA has CBASIC2, CP/M and Microsfot Z80 softcard in stock.

GENERAL LEDGER

Master File
Receipts and Disbursements
Balance sheet
Income Statement
Comprehensive Budget Analysis
Trial Balance and more

ACCOUNTS PAYABLE

Cash Requirements Report
Generates Purchase Orders
Automatic Posting
Prints Checks

INVENTORY MANAGEMENT

Comprehensive Status Reports
Inventory Alert Reports
Raw and Finished Goods
and more

ACCOUNTS RECEIVABLE

Complete Order Entry
Automatic Posting
File Aging (30/60/90 days)
Prints Invoices
Audit Trails

PAYROLL

Comprehensive Payroll Register
Prints Payroll Checks
Automatic Posting
Form 501, W2, 941

FIXED ASSETS ACCOUNTING

Complete Capital Goods Record
Option of 5 Depreciation Methods
IRS Depreciation Report

EXTRA: MAILING LIST PROGRAM

Features

Menu Driven: The entire system runs from a single master menu which accesses numerous subsidiary menus, when needed, to perform the full spectrum of business and accounting functions.

Self-Documenting: All the information needed to use the sytem is provided in an easy to self-study format.

Requirements: 48K CBASIC2
2 DISK DRIVES CP/M

Send $159 for the VISACCOUNT system

COMPUTER SERVICES CORPORATION of AMERICA
332 East 30th Street New York, New York 10016
Order Toll Free 1-800-221-2486 ext. 1591
Technical Number 1-212-685-0090

Name ______________________________

Address ______________________________

City/State/Zip______________________________

☐ Master Charge ☐ Visa ☐ American Express

No. ____________________ Expires ____________

Your System ______________________________

Disk Size ☐ 5¼" double density ☐ 8" single density

News & Views, continued...

Supersoft will upgrade their ADA package to a completely validated version through subsequent releases. The package will most likely sell for about $300. They also rumored to be developing 8086/8088, 68000 and Z8000 versions.......Microsoft and Digital Research are both rumored to be working on Visi-Calc look-alikes.

Remember When?

Do you remember when the Altair-8800 S-100 system came with a 1-Kbyte RAM card? With four cards you could run MITS 4-K Basic. Those were the days! Now, only six years later, you can get 256-Kbytes on one S-100 card and for a little more you can get a 9th parity bit with a parity-checker circuit. When will we see 1-Mbyte on a standard height (5") S-100 RAM card?

The new IEEE-696/S-100 standard specifies two board heights: standard 5", and a double height board 10" high. Several S-10 board manufacturers are planning to introduce products using this double height. Just imagine what you can put on a board of that size! Now, the question is—do I just leave the top off my mainframe, or do I get an extra-deep box? Oh well, it's nice to have an S-100 system that is always changing, and keeping up to the "state-of-the-art"! ■

Microsystems Bug

We sincerely regret that in the November/December 1981 issue, the Pascal software in Jon Bondy's article "Virtual Segment Procedures" was not in the correct order. Those readers who would like a correct print-out of the program should send a stamped (20¢), self-addressed #10 envelope to the author.

DUAL/68000

...Hard Power from the Soft Support of UNIX®

A new and powerful computer has been born... the System 83. The versatile UNIX* operating system pilots the System 83's raw power through a myriad of software such as "C", FORTRAN, PASCAL, BASIC, COBOL, and even Networking. Step into a bold new frontier with more power than you ever dreamed possible.

FEATURES:

□ UNIX V7 configured by UNISOFT**
□ Full IEEE 696/S-100 Compatability
□ MC68000 8Mhz Processor
□ 32-Bit Data Operations with 32-Bit Internal Registers
□ 16-Bit Data Transfer Operations
□ Memory Management Allows Concurrent Use of Mapped and Non-mapped Address space
□ Rugged Industrial Grade Components at all Levels
□ 16 Mb of Main Memory Directly Addressable
□ 7 Vectored Interrupt Levels
□ 192 Device-supplied Interrupts
□ 256 Kb of RAM with Parity Per Board Slot
□ Up to 3.2 Mb of RAM Per Cabinet

* UNIX is a trademark of Bell Laboratories and is supported on the DUAL System 83 by UNISOFT

**UNISOFT is a trademark of UNISOFT Corporation of Berkeley, CA.

DUAL

DUAL SYSTEMS CONTROL CORPORATION

system reliability/system integrity

720 Channing Way • Berkeley • CA 94710 • (415) 549-3854 • Telex: 172029 SPX

by Sol Libes

EDITOR'S PAGE

The CP/M Public Domain Software Libraries

I consider one of CP/M's most important advantages to be its huge public-domain software base. There are presently two organizations which provide this public-domain software at essentially the cost of the media, postage and handling. Together they provide over 100 volumes (each volume is an 8" single density floppy disk (containing well over 4,000 programs—some 20 Mbytes of software—that the contributors have put into the public domain. Most of the software is in source code form. There are languages, applications packages, utilities, games and much more.

The libraries are run by the CP/M User Group (CP/MUG) and Special Interest Group/M (SIG/M). The primary function of each group is the gathering, editing, cataloging, production and distribution of these disks. Both also have a printed catalog available. The CP/MUG is operated as an adjunct of Lifeboat Associates, an international distributor of commercial software. Lifeboat maintains the group with the assistance of the CACHE group (Chicago Area Computer Hobbyist Exchange). CACHE edits and catalogs the software and compiles each volume, while the CP/MUG collects the software and produces and distributes the disks. The SIG/M is operated jointly by the Amateur Computer Group of New Jersey (ACG-NJ) and the New York Amateur Computer Club (NYACC) which is based in New York City. These two clubs have a joint membership of close to 2,000, with most using CP/M-based systems. The SIG/M performs all of the functions of collecting, editing and distribution of their software.

The two groups have similar operating policies, distributing the disks to computer clubs who in turn are responsible for copying the software to supply their local area. Neither group is prepared to deal directly with individual users. For example, the SIG/M depends on a group of about a dozen hobbyist volunteers to do all the work on their own home systems. Hence, the SIG/M will furnish disks to individuals only if there is no distribution point convenient to the user. A list of the SIG/M distribution centers is included at the end of this article. These groups generally distribute both the CP/MUG and SIG/M software. The general policy followed by groups distributing the software is to charge $1 per disk when the disks are copied at meetings of the group. Furthermore, most of this software is maintained on-line on several hobbyist-run dial-in systems across the country. A caller using a modem and some appropriate file transfer software (e.g. the MODEM or MODEM7 programs in the CP/MUG and SIG/M libraries) can down-load the software directly into his/her system. In fact, this is the preferred method to overcome disk system incompatibilities when the user has a non-standard CP/M system. If the user does not find the software on-line, he can ask that the system operator (SYSOP) load the software onto disk for transfer at some future pre-arranged time.

Even if you do not transfer software from these on-line CP/M systems, it is interesting to read their bulletin boards as they often contain very useful information about users' experiences with CP/M, MP/M and microcomputers in general. A listing of these remote dial-in CP/M systems, and how to access them, will be found in the May/June 1981 issue of *Microsystems*. If you are interested in learning more about these software libraries, I would recommend that you first purchase a copy of their printed catalogs so you can see what software they have available. The CP/MUG library catalog is available from: Lifeline Publishing Corporation, 1651 Third Ave, New York, NY 10028. The catalog is $6 domestic, $11 foreign. Also, they publish a monthly twenty page newsletter which provides information on Lifeboat and CP/MUG software. The charge for the newsletter is $18/yr (U.S., Canada & Mexico), $40 elsewhere. The NYACC (New York Amateur Computer Club) publishes a 200 page catalog which contains the listings of both the SIG/M and CP/MUG libraries. They charge $10 for domstic orders and $13 for foreign. Order the catalog from: NYACC-CP/MUG, Box 106, Church Street Station, New York, NY 10008.

The SIG/M publishes an infrequent column which is carried on many of

At last! A "SCREEN HANDLER" for CBASIC2 programmers.

...source code to handle full screen input with multiple screens.

SCREENMASTER®

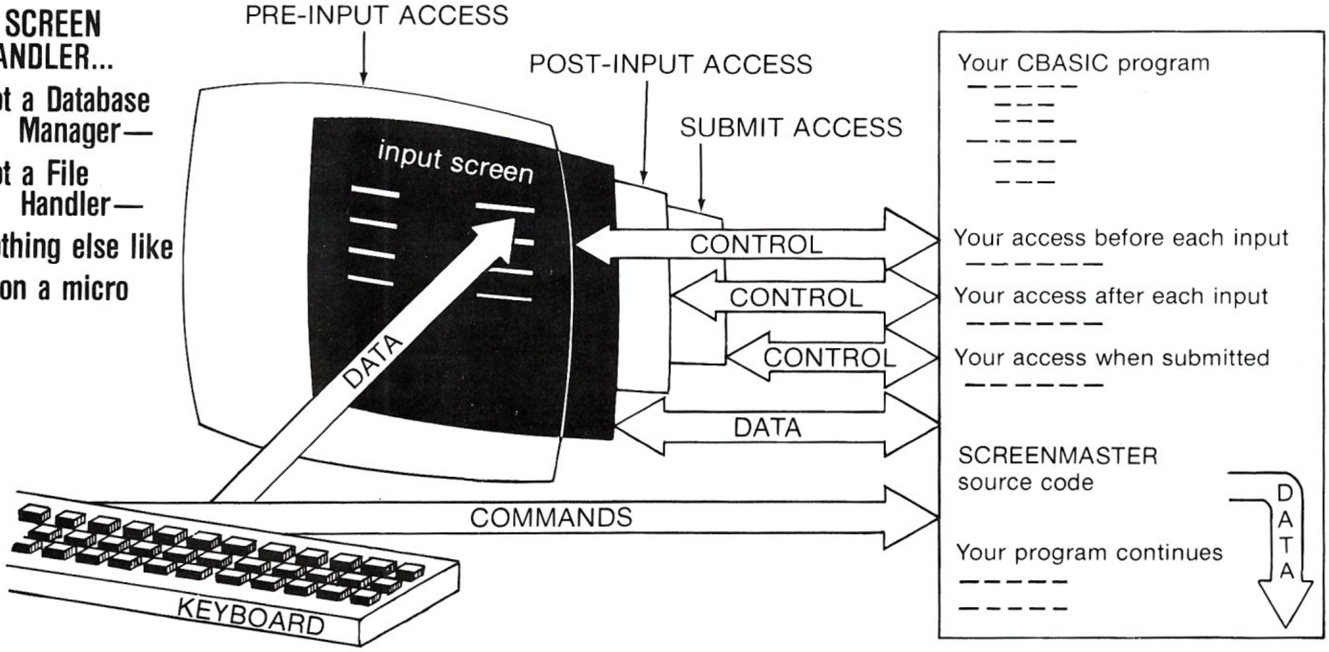
a program module that makes data entry a snap.

A SCREEN HANDLER...

not a Database Manager—

not a File Handler—

nothing else like it on a micro



- %include directly in program
- full screen input
- multiple screens
- user commands
- total programmer control in accesses
- terminal independent (>= 80 x 24)
- default editing with overriding
- screen maker utility
- screen tester utility
- run time terminal configurator utility
- separated programmer interface module
- easy to use • easy to install
- demo disk available
- extensive manual
- not a database manager or file handler
- runs with CP/M and CBASIC2

%INCLUDE DIRECTLY IN PROGRAM. SCREENMASTER is provided in source code and can be included directly in your program as if you had written it yourself. Total control of your program is thus assured.

FULL SCREEN INPUT WITH MULTIPLE SCREENS. User enters data on the formatted screen of your design. A single screen can be converted to multiple screens without program changes.

USER COMMANDS. User has commands like FORWARD (n fields), BACKWARD (n fields), GOTO (field m), NEXT (screen), PRIOR (screen), or PRINT (screen p). User commands may be abbreviated, implemented as special function keys, or individually disabled.

**CP/M a trademark of Digital Research, Inc. CBASIC2 a trademark of Compiler Systems, Inc.*

TOTAL PROGRAMMER CONTROL IN ACCESSES. Programmer is given access to SCREENMASTER operations in the exits—pre-input, post-input, and submit. The post-input exit, for example, lets you override SCREENMASTER editing, change any field, issue any command, and issue or override any return code.

UTILITIES INCLUDED. The SCREEN MAKER utility lets you develop and view your screen design interactively. The SCREEN TESTER utility lets you simulate the workings of your program by giving you interactive control in the exits. The TERMINAL CONFIGURATOR utility writes your terminal characteristics to a file that is read in when your program runs. Your terminal can be easily added to the list of available terminals.

EASY TO USE. Even a novice programmer can use it. Issue a simple GOSUB. When it returns your validated data is available for your further use. No programming is necessary. But if you need to do something special—like search a file—you can do it yourself in the exits.

FLEXIBLE. If you're an experienced programmer you'll appreciate the lengths we've gone to to enable you to interface and modify SCREENMASTER. Initialization is a separate, well documented module. All communication with your data and screen formats is via functions in another separate, well documented module. Even the guts of SCREENMASTER is well documented, allowing some simple modifications.

Send your check for $195 to **MARKETING ESSENTIALS, INC.,** 206 Mosher Avenue, Woodmere, N.Y. 11598. Or call (516) 569-4533/(800) 645-2622. Manual available separately for $25, applicable towards the full purchase price. A demo disk is available for an additional $10.

MARKETING ESSENTIALS, INC.

206 Mosher Avenue/Woodmere, N.Y. 11598 Telephone (516) 569-4533/(800) 645-2622

Editor's Page, continued...

the remote dial-in CP/M systems and can therefore be read at no charge. A few of the systems even carry the complete catalog on-line. However, I recommend that you purchase a copy, as it is professionally printed and would take a very long time to down-load the catalog information. The S!G/M column is also printed in the newsletters of the NYACC and ACG-NJ.

The costs of the disks are:

CP/MUG: $8/disk USA, Canada & Mexico $12/disk overseas.

SIG/M: $6/disk USA, Canada & Mexico International add $4.

If the the SIG/M disks are copied at meetings of the ACG-NJ or NYACC, a donation of $1/copy is asked for. Savings on postage and handling are available from the SIG/M if more than one disk is ordered. When dealing with these groups you should allow 3-5 weeks for them to ship. The SIG/M disks can be ordered from: SIG/M, Box 97, Iselin, NJ 08830. The CP/MUG disks can be ordered from: CP/MUG, 1651 Third Ave, New York, NY 10028.

Note that both groups furnish their disks also for North Star systems (DD or SD). When using DD, one volume is stored on two disks, for SD one volume is stored on four disks. Lastly, the SIG/M can also furnish disks in Apple (single density), Cromemco (5" & 8"), Micropolis Mod-II double density 5" and TRS-80-I/II/III forms. ■

SIG/M Software Distribution Groups

Alaska			
	Anchorage	99501	John Evans 618 "N" Street
Arizona			
	Dewey	86327	Thomas Oliver, Blue Hills CPMUG Blue Hills Rt
California			
	Escondido	92025	Richard Mason, (714)746-4832 San Diego Computer Society 1037 Park Hill Lane
	Imperial Valley	92251	J.R. Pendley Imperial Valley Informal Computerists P.O. Box 158
	Menlo Park		Gordon French, (415)325-4209 Homebrew Computer Club 614 18th Avenue
	Mill Valley	94941	Jim Ayers, CBBS (415)383-0473 Apple CPMUG of Small Computer Users of Marin 301 Poplar St.
	Mt View	94040	Bruce Kendall, 100 BUSS 334 A. Camille Ct
	Nevada City	95959	Bob Cobler Motherload Computer Club
	Sacramento	95823	Charlie Foster, (916)392-2789 Pascal/Z Users Group SIG/M - Western Coast Region 7962 Center Parkway
	Sacramento	95816	John Moorhead, (916)758-2495 Sacramento Microcomputer Users Group P.O. Box 161513
	San Bernardino	92412	Bob Massey, CP/M Users of Compuserve Compuserve P.O. Box 6212
	Simi Valley	93065	Kelley Smith, CBBS (805)527-9321 CP/M-NET, (805)527-0518 3055 Waco Ave
	Sunnyvale	94086	Samuel Daniel, Silicon Valley CPMUG S.D.C. 500 Macara Ave
	Temple City	91780	Howard Stone, Temple City Computer Hobbyists P.O. Box 572
Colorado			
	Littleton	80123	Larry Thiel, Denver Amateur Computer Society W. Capri Drive
Florida			
	Hollywood	33024	Ralph Fernandez, (305)963-7893 South Florida Computer Club 1231 NW 72 Avenue

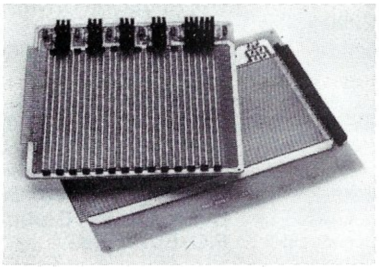
Largo	33540	Richard Tremmel, Coastal Computer Club 10585 119th St N.
Titusville	32780	Michael Seay Space Coast Microcomputer Club 995 Luna Terrace
Georgia		
Atlanta	30338	Lewis Mosley, Atlanta Computer Society P.O. Box 88771 Atlanta, GA 30338
Hawaii		
Honolulu	96826	Victor Mori, (808)955-6683 2525 So. King St.
Illinois		
Glen Ellyn	60137	Stanley Hanson 192 East Road
Lake Forest	60045	Calamity Cliffs Computer Center P.O. Box 392
Indiana		
Ft Wayne	46805	Geoffrey Priest, (219)423-1571 Prestige Marketing Corp. 909 N. Coliseum Blvd.
Indianapolis	46219	George Wilson, Indianapolis Small Systems Group - CPMUG 6808 E. 21st St.
Massachusetts		
Bedford	01730	Dave Mitton, CBBS (617)864-3819 New England Computer Society P.O. Box 198
Littleton	01460	Neil Rosenburg, S-100 Club 2 Deer Run
Michigan		
Grosse Pointe	48230	Dave Hardy, (313)885-0506 Technical CBBS, (313)846-6127 736 Notre Dame, (313)846-8000
Royal Oak		Keith Peterson, CBBS (313)588-0754
Westland		Ron Fowler, CBBS (313)729-1905
Missouri		
St Louis	63131	John Taylor St. Louis Area Computer Club 2009 N. Geyer Road
New Jersey		
Iselin	08830	SIG/M-Main HQ., CBBS (201)272-1874 P.O. Box 97
Riverside		Kevin O'Connell, (609)461-4351
Pinebrook		Ray Glueck, (201)227-5361 Micropolis CPMUG of ACG-NJ
Cranford		Bruce Ratoff, CBBS (201)272-1874 SIG/M - CBBS, (201)272-1793 26 Broad St.
Clifton	07001	William "Bill" Chin (201)778-5140 SIG/M - Operations/Editor/VP.ACG-NJ 177 Hadley Ave
Chatham		Randy Reitz, (201)635-5642 N*UG ACG-NJ
Cliffside Park	07010	Steve Leon, H (201)886-1658 TRS-80 SIG/M Librarian, (212)488-7677 200 Winston Drive
Dover		Marty Nichols, (201)361-7180
Piscataway		Steve Toth, (201)968-7498 Apple Users Group ACG-NJ
Haddonfield	08033	Michael Sulllivan, (609)795-5607 Financial Software 54 Grove St

Editor's Page, continued...

Williamstown	08094	Mike Mitelski, (609)629-0568 Philadelphia Area Computer Society 577A Prossen Ave, RD 3
Ocean	07712	Thomas Lindsay, (201)946-9874 Seashore Computer Club 1108 Pollack Ave
Pennsauken	08109	Dean Kelchner, (609)663-2842 RCA Microcomputer Club 5340 Peabody Ave
New York		
Flushing	11355	Henry Kee, (212)538-3202 SIG/M - Editor New York Amateur Computer Club 42-24 Colden St
Henrietta	14467	William Nixon, Rochester RAMS 115 Clooney Drive
Dix Hills	11746	Gary Fishkin, Long Island Computer Assoc 82 Dix Highway
Putnam Valley	10579	Bryan Lewis, NY Sorcerers Users Group RD 3, Florence Road
Peekskill	10566	Helmut Ripke, (914)739-3754 Taconic Computer Club 935 Frost Court
Ohio		
Copley	44321	Charles Lewis, C.D.G. 379 S. Hametown Road
Dayton	45432	Richard Conn 4927 Woodward Park Drive
Oregon		
Portland	97212	Carl Townsend, (503)282-5835 Portland Computer club 4110 N.E. Alameda
Pennsylvania		
Orefield	18069	William Earnest, (215)398-1634 Lehigh Valley Computer Group RD 1, Box 830, CBBS (215)398-3937
Whitehall	18052	Stan Rinkunas 352 Sumner Ave
Sewickley	15143	Hassan El-Zayyat 256 Bank St
Erie	16509	Charles Fisher, Erie Computer Club 5520 Herman Drive
New Hope		Jim Woolsey, (215)862-5806 Delaware Valley Computer Club 6 Stone Hill
Texas		
Plano	75075	Fred Pfafman, (214)596-5034 2320 Heather Hill Lane
Houston	77057	Jerry Ambroze Ambroze & Associates 2168 Augusta
Missouri City	77489	Al Whitney, (713)438-1750 SIG/M - South Central 2003 Hammerwood
Virginia		
McLean	22101	Robert Teeter, (703)356-1745 Metro Washington CPMUG 6410 Furlong Road

DEVELOPMENT TOOLS

Simplify your S-100 product development with our 10"x10" double-height wirewrap prototype and extender cards



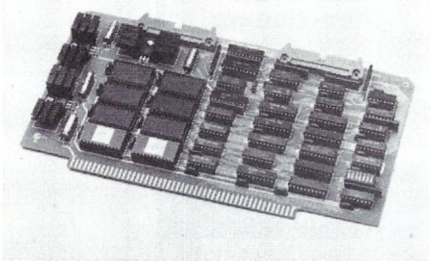
10" HIGH PROTOTYPE BOARD

- Regulators provide 5V @ 4A, ±12V @ 1A
- Accepts over 120 IC's plus 3M-type connectors
- Heavy power distribution with 28 distributed 0.1 ufd caps
- Price $119 ea., 3 for $298

13" HIGH EXTENDER CARD

- Scope probe hanger on each trace
- Ground trace between signal lines
- Heavy power traces with removable jumpers for current measurement
- Price $68 ea.

No downloading – No trial PROM burning. This port-addressed RAM on your S-100 host is the ROM of your target system



WORD/BYTE WIDE ROM SIMULATOR

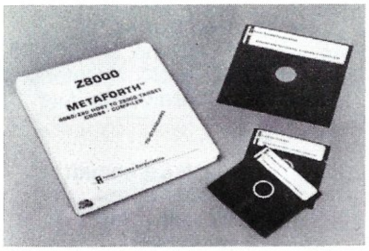
- Simulates 16K bytes of memory (8K bytes for 2708 and 2758)
- Simulates 2708, 2758, 2516, 2716, 2532, 2732, 2564 and 2764 PROMS
- The simulated memory may be either byte or 16-bit word organized
- No S-100 memory is needed to hold ROM data
- Driver program verifies simulated PROM contents
- Price $495 each

Learn the fastest, most compact and machine independent language for microprocessors – FORTH

— LEARN FORTH NOW —

Take our intensive one-week FORTH workshop. Enrollment is limited to 8 students.
The course is offered once a month at $395 per student.

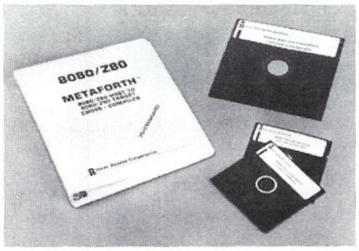
Now you can develop code for a target Z8000 system on your 8080/Z80 CP/M® or Cromemco CDOS® based system



Z8000 FORTH DEVELOPMENT SYSTEM

- Works with the Zilog Z8000 Development Module and may be easily tailored for use with other Z8000 target machines
- Includes Z8002 assembler
- Can produce code without header and link words for up to 30% space savings
- can produce ROMable code
- 79 Standard FORTH
- Price $1450

Develop FORTH code for any target 8080/Z80 system on your current 8080/Z80 or Cromemco CDOS based system



8080/Z80 METAFORTH CROSS-COMPILER

- Produces code that may be downloaded to any Z80 or 8080 processor
- Includes 8080 and Z80 assemblers
- Can produce code without headers and link words for up to 30% space savings
- Can produce ROMable code
- 79 Standard FORTH
- Price $450

Inner Access Corporation

P.O. BOX 888 • BELMONT, CALIFORNIA 94002 • (415) 591-8295

BANKAMERICARD VISA

WE HONOR master charge THE INTERBANK CARD

Editor's Page, continued...

Fredricksburg	22401	Jack Williams Microcomputer Investors Assoc 902 Anderson Drive
Charlottesville	22901	A.C. Weaver School Of Engineering Thorton Hall University of Virginia
Alexandria	22307	William Higgs, (703)765-8043 Washington Area TRS-80 Users 1715 Hollinwood Drive
Grafton	23692	David Holmes, (804)898-5913 Digital Interest Group in Tidewater P.O. Box 1708
Washington		
Bellevue	98004	George Clark, (206)454-6826 Northwest Computer Society 307 108th Ave S.E.
Seattle	98117	David Rabbers Ballard CP/M Users Group 6551 16th Ave NW
Wisconsin		
Milwaukee	53213	Rick Martinek, CBBS (414)774-2683 Ricks Computer Center, (414)774-8445 5903 West Parkhill
INTERNATIONAL		
Australia		
Mytrle Bank	5064	Anthony Beresford CPMUG of South Australia 46 Cross Road
Canada		
Regina	S4V0V7	Bob Stek, R.O.M.S. 19 Mayfield Road
Vancouver	V6B3A0	Dave Bowerman P.O. Box 4476
Toronto	M5G1X6	R.J. Dunn, (416)592-5788 CP/M UG of Ontario HYDRO 700 University Ave
Netherlands		
Rotterdam		Hank Berkhoudt CP/M Groep Hesselskamp 4 3085 SM
Venezuela		
Caracas	1061A	Hans Stauffer Caracas Computer Club Apartado 66394
		Hans Stauffer c/o Eng. Eduardo Stiassnis M105 Jet International Airport P.O. Box 592857 Miami, FL 33159
Singapore		
Singapore	1543	Alex Chan 745 Mountbatten Road
Singapore	0104	Naresh Kapoor Patel Computer Systems PTE 2705-8 OCBC Centre Chulia St

LETTERS TO THE EDITOR

'Clean Air' for the North Star

Dear Editor:

The Vestal Press, Ltd. is a small specialty book publishing and distribution business, specializing in mechanical hobby books mainly in the field of antiquities. Books on player pianos, music boxes, merry-go-round organs, gambling machines and the like constitute the bill of fare.

In 1980 the work load got to the point where a small computer was indicated, so a system consisting of a 48K Dual Density Quad machine together with an Anadex printer and a Hazeltine 1500 terminal was installed by Micro World Inc., of Vestal, New York. This firm wrote some very sophisticated software to handle order processing for about 750 inventoried items, and an active mailing list of around 8500 customers.

The environment in which the firm works is simply not as free of dust as one might like, but economic considerations rule out a completely dust-free room just for computer operations. In spite of this, everything worked well for the first several months after the installation.

Then we started to have rash of disk failures, all of which were quite obviously caused by dust on the read-write heads of the disk drives; three or four a week became common, and something obviously had to be done. Back-up disks provided a way of staying in business, of course, but much time was lost in following such procedures.

Our dealer, Wayne Kashinsky, suggested that maybe reversing the muffin ventilating fan might help, so we did this and went one step better by constructing a filter holder on the outside of the machinee. The holder is made of sheet metal, with the large holes (the same diameter as the fan) being turned on a lathe, and the tube being rolled section of the same sheet metal. The entire assembly was soldered together and then painted, and before being screwed to the back of the computer, a rubber-cork gasket was fitted.

Since the installation, our disk failures have dropped practically to zero, so obviously the idea has served us well.

We didn't make any big engineering research effort to decide whether the air being heated by the power supply might cause trouble as it passed through the drives and the rest of "Winifred" (that's what we call her), although we did make a casual call to a telephone-answering technician at North Star. He rendered an unofficial opinion that the idea probably wouldn't hurt anything, and might well help.

So the lesson here is that forcing air throughout your North Star instead of sucking air into it and then out seems to help if you have to operate in an area which isn't as clean as you'd like. If you have experienced disk failures, we heartily recommend that you give this idea a try!

Harvey N. Roehl
The Vestal Press Ltd.
Vestal, NY

No Help In Sight

Dear Editor:

My business uses a Cromemco Z2 computer and I have had three years of good service with it. However, this summer a piece broke in the PerSci disk drive and I have been unable to get it fixed. I have written PerSci, Cromemco, an advertiser in the Cromemco User's Group Newsletter, as well as having a local dealer try to get the part for me.

Cromemco did answer my letter after about six weeks, but offered no help. PerSci has yet to be heard from. The advertiser answered promptly, but did not sell parts. The local dealer drew a blank with PerSci, also.

So I limp along with a single drive, not knowing where to turn. Cromemco is doing well, probably best of all the S-100 companies, and I see that PerSci has a new prestigious ad out. But can they compete with IBM with no spares support?

Malcolm Gillis
President
MEGA Corporation

Turnkey S-100 Systems For Schools

Dear Editor:

A brief examination of any of the current computer magazines (for example, the October 1981 *Creative Computing*) will reveal that the S-100/CP/M community is about to lose the educational market completely—or may have already lost it. This turn of events is unfortunate from the viewpoint of the manufacturers, because the market is a very large one. Even worse, it bodes ill for the future. A person who has become familiar with an Apple, PET, Atari, TRS-80 etc. as a student is very likely to select the same brand for personal or professional work later on.

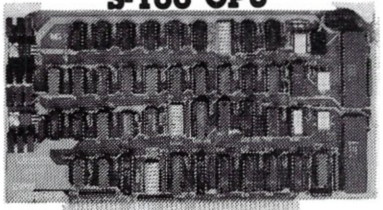
The cause of the problem is easy to locate: *lack of a suitable, reasonably-priced TURNKEY S-100 system* on the market. The situation has more to do with system configuration than with price. A suitable system configuration is as follows:

- *Dual disk, 8- to 12-slot mainframe with main and disk power supplies.* (Exam-

PRIORITY ONE ELECTRONICS

1 ONE

S-100 CPU



CPU-Z - GODBOUT

2/4 MHZ Z80 CPU 24 Bit Addressing

GBT 160A	A&T	$199.00
GBT 160C	CSC 3-6 MHZ	$375.00

DUAL PROCESSOR 8085-8088 - GODBOUT

6 or 8 MHz Provides true 16 Bit Power with a standard 8 bit S-100 bus.

GBT 1612A	A&T . . .6 MHZ	$399.00
GBT 1612C	CSC . . . 8 MHZ	$498.00

SOLID STATE DISK DRIVE, 3500% FASTER!

Not Really, But the Next Best Thing For Godbout 8085/88 Users. Call For Details on M-Drive. See Page 340 of November **BYTE**

GBT MD 128K	....	$1550.00
GBT MD 256K	....	$3,000.00

2810 Z80 CPU-CA. COMP. SYST.

2/4 MHz Z80A CPU with RS232C Serial I/O Port complete with Monitor PROM for 2422 Disk Controller

CCS 2810A	A&T......................	$280.00

CB2 Z80 CPU - S.S.M.

2/4 MHZ will accept 2716, or 2732, or RAM RUN/STOP and single step switches

SSMCB2K	Kit	$260.00
SSMCB2A	A&T	$310.00
SSMZ80M	SSMZ80 Monitor . . .	$89.00

CBIA 8080 CPU - S.S.M.

8080 CPU, 1K RAM, Holds 1 2708, 1 8 Bit parallel input port.

SSMCB1A	Kit	$183.00
SSMCB1A	A&T	$225.00
SSM8080M	SM 8080 Monitor	$59.00

S-100 I/O BOARDS

SYSTEM SUPPORT 1 - GODBOUT

Serial port (software prog baud), 4K EPROM OR RAM provision, 15 levels of interrupt, real time clock, optional math processor

PART NO.	DESCRIPTION	LIST PRICE	OUR PRICE
GBT162A	Assembled & Tested	$39.00	$360.00
GBT162C	CSC...............	$495.00	$460.00
GBT8231	Math Chip		$195.00
GBT8232	Math Chip		$195.00
GBT162AM1	A&T with 8231 Math Chip		$555.00
GBT162CM1	CSC with 8231 Math Chip		$655.00
GBT162AM2	A&T with 8232 Math Chip		$555.00
GBT162CM2	CSC with 8232 Math Chip		$655.00

MPX CHANNEL BOARD - GODBOUT

I/O Multiplexer, using 8085a-2 cpu on board

GBT166A	Assembled & Tested	$495.00	$450.00
GBT166C	CSC.............	$595.00	$550.00

INTERFACER I - GODBOUT

Two Serial I/O

GBT133A	A&T..............	$249.00	$219.00
GBT133C	CSC.............	$324.00	$298.00

INTERFACER II - GODBOUT

Three parallel, one serial I/O board

GBT150A	A&T..............	$249.00	$219.00
GBT150C	CSC.............	$324.00	$289.00

INTERFACER III - GODBOUT

Eight channel multi-use serial I/O board

GBT1748A	Assembled & Tested..	$699.00	$629.00
GBT1748C	CSC 200 hr. Burn In Test................	$849.00	$629.00

INTERFACER 3 WITH 5 SERIAL PORTS

GBT1745A	Assembled & Tested	$599.00	$553.00
GBT1745C	CSC 200 hr. Burn In	$699.00 Test	$629.00

MULTI I/O - MORROW DESIGNS

Three Serial, Two parallel

MDSMB3200	A&T..............	$329.00	$309.00

SWITCHBOARD-MORROW DESIGNS

Two serial I/O, four parallel I/O, one status port, one strobe port

MDSSB2411		$259.00	$239.00

I/O4 - SSM

Two serial I/O, two parallel I/O

SSMI04K Kit			$210.00
SSMI04A A&T		$290.00	$260.00

2710 4 PORT SERIAL - CCS

4 Full handshaking RS232 ports and optional 2K ROM

CCS271001	A & T	$340.00	$310.00

2718 2 SERIAL & 2 PARALLEL - CCS

2 RS232 C ports, 2 8 bit parallel ports, & optional 2K ROM

CCS271801	A & T.............	$360.00	$325.00

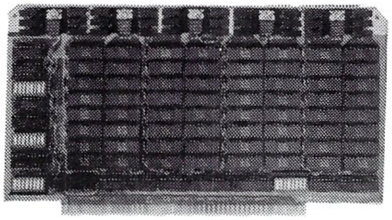
2720 4 PORT PARALLEL - CCS

4 8 bit parallel ports and optional 2K ROM

CCS272001	A & T.............	$250.00	$225.00

S-100 10 MHZ STATIC RAM

NEW LOW PRICES!



32K STATIC RAM - GODBOUT

RAM 20 10 MHZ, 4Kbyte block disable, bank or 24 bit addressings available 8, 16, 24 or 32K

PART NO.	DESCRIPTION	LIST PRICE	OUR PRICE
GBT164AA8	8K A&T	$210.00	$190.00
GBT164AC8	8K CSC	$280.00	$260.00
GBT164AA16	16KA&T	$285.00	$260.00
GBT164AC16	16K CSC	$355.00	$325.00
GBT164AA24	24K A&T	$355.00	$325.00
GBT164AC24	24K CSC	$425.00	$385.00
GBT164AA32	32K A&T	$425.00	$385.00
GBT164AC32	32K CSC	$495.00	$450.00

64K STATIC RAM - GODBOUT

RAM 17, 10 MHZ, 2 Watt, DMA Compatable 24 Bit Addressing

GBT175A48	48K A&T	$650.00	$619.00
GBT175C48	48K CSC 200hr.	$750.00	$710.00
GBT175A64	64K A&T	$795.00	$755.00
GBT175C64	64K CSC 200hr.	$895.00	$850.00

NEW! 32K x 16 BIT STATIC RAM - GODBOUT

RAM 16 10 MHZ, 32K x 16 or 64K x 8 IEEE/696 16 BIT 2 Watt, 24 Bit Addressing

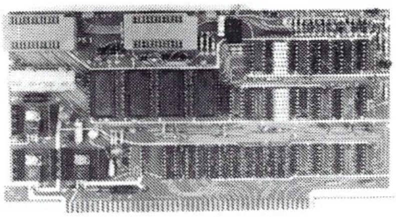
GBT180A	64K A&T	$895.00	$850.00
GBT180C	64K CSC	$995.00	$945.00

NEW! 128K STATIC RAM - GODBOUT

RAM 21 10MHZ 128K x 8 or 64K x 16

IEEE/696 8 or 16 Bit 1.2 Amps 24 Bit Addressing

GBT167A	128K A&T	$1695.00	$1610.00
GBT167C	128K CSC	$1895.00	$1795.00



S-100 ROM

PBI PROM PROGRAMMER - SSM

Programs 2708 or 2716's, operates as a 4K/8K EPROM BOARD AS WELL.

SSMPB1K	Kit		$179.00
SSMPB1A	Assembled & Tested	$265.00	$220.00

ECONOROM 2708 - GODBOUT

16K x 8eprom Board using 2708, Power on jump to any 256 byte

GBT125A	Assembled & Tested	$135.00	$120.00
GBT125C	CSC	$195.00	$175.00

S-100 VIDEO BOARDS

SPECTRUM - GODBOUT

Color Graphics board with Parallel I/O

GBT144A	Assembled &Tested	$399.00	$349.00
GBT144C	CSC	$449.00	$399.00
GBT2D	Sublogic Universal Graphics Interpreter Software		$35.00

VB - 3 S.S.M.

80 x 25 or 50 character video display Memory Mapped, Parallel Keyboard port

SSMVB3K24	80x24 Kit		$425.00
SSMVB3A24	80x24 A&T	$499.00	$440.00
SSMVB3UP	80x50 Line Upgrade		$ 39.00

VB2-S.S.M.

I/O Mapped Video Board, with Parallel Keyboard port 64 x 16

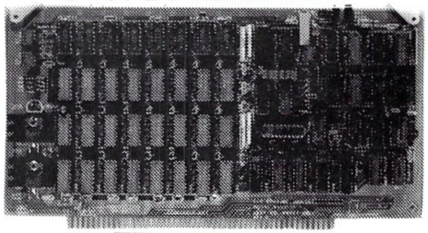
SSMVB2K	Kit		$199.00
SSMVB2A	Assembled & Tested	$269.00	$229.00

VBIC - S.S.M.

Memory Mapped Video Board 64x16 character display or 64x16 graphics display

SSMVB1K	Kit		$199.00
SSMVB1A	Assembled & Tested	$269.00	$229.00

S-100 DYNAMIC RAM

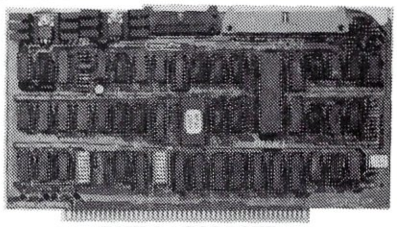


THE EXPANDABLE 1
PRIORITY 1 ELECTRONICS

THE EXPANDABLE 1™ 64 K Dynamic Ram board provides your S-100 system with 64K of reliable, high-speed dynamic RAM. Compatable with most of the major S-100 systems on the market, including those with front panels, it supports DMA operations and requires no Wait states with current microprocessors. ● User expandale from 16 to 64K ● Supports DMA ● Designed to IEEE proposed S-100 bus standards ● 2 or 4 MHz operation ● Operates with either an 8080 or Z-80 based S-100 system, providing processor-transparent refreshes with both ● Supports IMSAI-type front panels ● Jumper-selectable Phantom input ● Uses Popular 4116 RAMS ● All ICs in sockets ● Any 16K block can be made bank-independent ● Fully buffered address and data lines ● Fail-safe refresh circuitry for extended Wait states ● Board configuration with reliable, easy-to-configure Berg jumpers

PRIEXP116	16K Assembled & Tested	$299.00
PRIEXP132	32K Assembled & Tested	$339.00
PRIEXP148	48K Assembled & Tested	$379.00
PRIEXP164	64K Assembled & Tested	$409.00

S-100 DISK CONTROLLERS



DISK 1 - GODBOUT

FAST DMA, Soft Sector, Controls 8" or 5¼", single or double density OUR BEST!

		LIST PRICE	OUR PRICE
GBT171A	Assembled & Tested	$495.00	$450.00
GBT171C	CSC.............	$595.00	$555.00
GBTCPM80*	CP/M 2.2 for Z80/8085 with manuals & BIOS 8" S/D disk		$175.00
GBTOAS8S	Oasis 8 bit single user 8" S/D disk		$500.00
GBTOAS8M	Oasis 8 bit multi-user. 8" S/D disk		$850.00

2422A - CA. COMP. SYST.

I/O Mapped, controls 8" or 5¼" single or double density A&T with CPM 2.2 8" S.D.

CCS2422A		$475.00	$375.00

DISK JOCKEY 2D - MORROW

I/O Mapped, controls 8", single or double density, serial I/O

MDSDJ2208	A&T with CP/M 2.2	$399.00	$375.00

S-100 DISK SUBSYSTEMS

DISCUS SINGLE SIDED MORROW

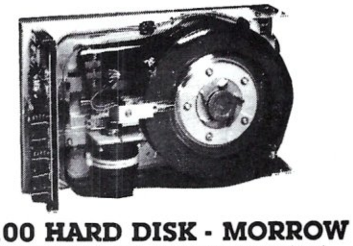
8" DBL Density drives with cabinet, power supply controller, with CP/M 2.2 and Microsoft Basic

MDSF1218	Single Drive System.	1095.00	$950.00
MDSF1228	Dual Drive System	$1875.00	$1598.00

DISCUS DOUBLE SIDED - MORROW

8" DBL Density/sided drives with cabinet Power supply controller, with CP/M 2.2 and Microsoft Basic

MDSF2218	Single Drive System	1395.00	$1250.00
MDSF2228	Dual Drive System	$2495.00	$2050.00



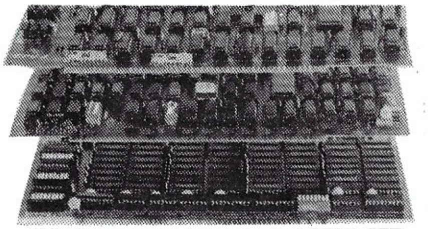
S-100 HARD DISK - MORROW

8" 10 & 20MB .14" - 26MB formatted hard disk complete with cainet, P.S., Controller, CPM 2.2 and Microsoft Basic

		LIST PRICE	SALE PRICE
MDSM10S	10 MB	$3695.00	$2950.00
MDSM20S	20 MB	$4795.00	$3825.00
MDSM26S	26 MB	$4495.00	$3495.00

PRIORITY ONE ELECTRONICS

S-100 SYSTEMS



"LITTLE 8" Z80 SYSTEM STARTER SET GODBOUT

CPU Z: *A 4MHz Z80 A-based 8-bit workhorse CPU board that includes all the standard features plus many of the convenience options. Meets all IEEE 696/S-100 specifications, including timing.*

DISK 1 DMA High Performance Disk Controller: *disk controllers don't have to be your system's bottleneck! The DISK 1 is lightning fast thanks to properly implemented DMA (with arbitration) and transfer that is independent of CPU speed.*

RAM 20 32K High Speed Static RAM. *This board has it all Operates at speeds up to 10MHz, ultra-low power consumption, IEEE 696/S-100 extended addressing protocol, bank select and flawless DMA.*

CP/M 2.2; *The de facto standard of 8-bit operating systems ready to load and go!*

ANOTHER PRIORITY 1 EXCLUSIVE!

We went to GODBOUT and made a special buy on the nucleous of the best S-100 Z80A systems ever.

LOOK AT WHAT YOU GET:

1 **GBT160A** *2/4 MHz Z80 CPU*	**$295.00**
1 **GBT164A32** *32K 10MHz Static Ram*	**$425.00**
1 **GBT171A** *DMA Disk Controllers*	**$495.00**
1 **GBTCPM80** *CP/M 2.2*	**$175.00**
IT ALL ADDS UP TO	**$1039.00**

TOTAL PACKAGE PRICE ONLY $1095.00
ORDER NO. PDBGBTSG

SUPERSIXTEEN — GODBOUT
LOOK WHAT $3495.00 WILL BUY!
WHY WAIT ANY LONGER?
HERE IS WHAT EACH PACKAGE INCLUDES:

GBT1612A *6 MHz 8085/8088 Dual Processor Board*
GBT171A *High Speed DMA Disk Controller*
GBT162A *System Support 1 Multi Function Board*
GBT133A *Interfacer 1 Dual Serial I/O*
128K 10MHz *Low Power Static Ram*
CP/M 86 *16 Bit Operating System Ready to Load & Go*
Cables and Documentation *Three interfacer cables one disk I/O cable, complete documentator for all hardware, and manuals for both CP/M operating systems.*
Compu Pro's *famous* **1 Year** *limited warranty.*

Now to the best part of all. If purchased separately, these quality components would list for $4,344.00. **BUT SuperSixteen's low package price is an amazing $3,495.00. Yo save $849.00!** *(For boards qualified under the Certified System Component high-reliability program - with extended 2 year warranty, 200 hour burn-in and 8MHz processors - add $600 to the package price.*

Sh. Wt. 15 lbs.

PDBGBTSJ	*SuperSixteen A&T*	**$3495.00**
PDBGBTSK	*SuperSixteen CSC*	**$4095.00**

S-100 SOFTWARE

PRIORITY 1 is pleased to offer the finest in industry standard software. All software is supplied on 8" Single Density IBM 3740 CP/M compatible disketts. All software is sold "AS IS" and is non-returnable. If you have questions about the software for your application, order the manual first.

CCS803 *CP/M Version 2.2 Microcompouter Control Program*	**$150.00**
CCS2301 *MAC-CP/M Macro Assembler*	**$90.00**
CCS2401 *SID-CP/M Symbolic Instruction Debugger*	**$75.00**
CCS2501 *TEX-CP/M Text Formatter*	**$75.00**
CCS2601 *DESPOOL-CP/M Background Print Utility*	**$50.00**

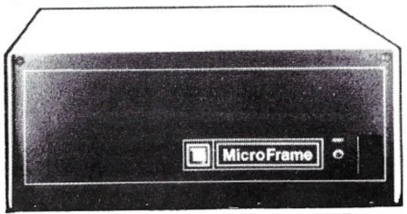
CP/M, MAC, SID, TEX, and DESPOOL are registered trademarks of Digital Research

PART NO.	DESCRIPTION	LIST PRICE	OUR PRICE
CCS401	*C-BASIC-2 Interp*	*$150.00*	$139.00
CCS401M	*Manual*		$ 32.00
CCS1101	*FMS-80 by Systems Plus*	*$995.00*	$895.00
CCS1101M	*Manual*		$ 70.00

GRAHAM-DORIAN ACCOUNTING

CCS1301	*General Ledger*	*$820.00*	$750.00
$CCS1301M	*Manual*		$ 50.00
CCS1501	*Accounts Receivable*	*$820.00*	$750.00
CCS151M	*Manual*		$50.00
CCS1401	*Accounts Payable*	*$820.00*	$750.00
CCS1401M	*Manual*		$ 50.00
CCS1701	*Inventory II*	*$820.00*	$750.00
CCS1701M	*Manual*		$ 50.00
CCS1601	*Payroll II*	*$555.00*	$495.00
CCS1601M	*Manual*		$ 50.00
CCS20001	*Job Costing*	*$820.00*	$750.00
CCS2001M	*Manual*		$ 50.00
CCS2701	*Order Entry/Invoice*	*$820.00*	$750.00
CCS2701M	*Manual*		$ 50.00
MEDICAL PRACTIC PATIENT BILLING			
CCS1801	*15 Programs*	*$820.00*	$750.00
CCS1801M	*Manual*		$ 50.00
DENTAL PRACTICE PATIENT BILLING			
CCS1901	*14 Programs*	*$820.00*	$750.00
CCS1901M	*Manual*		$ 50.00

S-100 MAINFRAMES



S-100 MICROFRAME - TEI

110V 60HZ CVT Mainframes, the best money can buy!
12 Slot ±8V 17A ±16V @ 2A
22 Slot ±8v @ 30A ±16V @ 4A

TEI has announced a 5 - 8% Price Increase Feb 1 - Hurry!

			OUR	PRICE
		LIST PRICE	1-9	10-24
TEIMCS 112	12 Slot Desk	$685.00	$615.00	$570.00
TEIMCS 122	*22 Slot Desk*	*$825.00*	$760.00	$705.00
TEIRM 12	*12Slot Rackmnt*	*$725.00*	$720.00	$619.00
TEIRM 22	*22 Slot Rackmnt*	*$875.00*	$850.00	$750.00

Shipping Weight: On 12 Slot Mainframes 45 lbs.
On 22 Slot Mainframes 55 lbs.

S-100 FRAMES 2 - 5" DISK CUTOUTS - TEI

±8V @ 17±16V @ 2A +12V @ 1.2A, Internal Cables

			1-9	10-24
TEITF12	*12 Slot desk*	*$675.00*	$625.00	$580.00
TEIRD12	*12 Slot Rackmnt*	*$795.00*	$715.00	$665.00

Shipping Weight: On 12 SlotDesk 40 lbs.
On 12 Slot Rackmount 45 lbs.

DUAL 8" DISK DRIVE CHASSIS - TEI

For Shugart 800/801R or 850/851R withinternal power cables provided
+24V @ 1.5A +5V @ 1.0A - 5V @ .25A

			1-9	10-24
TEIDFDO	*Desk Top*	*$535*	$485	$455
TEIRFDO	*Rack Mount*	*$720*	$670	$630
PDBDFDOS1	*DFDO with 1 Shugart 801R*			$970.00
PDBFDOS1	*RFDO with 2 Shugart 801Rs*			$1375.00
PDBRFDOS1	*RFDO with 1 Shugart 801R*			$1095.00
PDBRFDOS2	*RFDO with 2 Shugart 801Rs*			$1495.00
PRI50PGCE2	*Internal Data Cable .50 pin plug connector to 2 Card Edge.*			$34.95

Due to UPS shipping regulations, disk drives will be shipped separately from the cabinet. Don't forget to include shipping for each drive. (Shipping Wt. 16 lbs., each)

CALL FOR NEW TEI PRICES FEBRUARY 1st.

S-100 MAINFRAME - GODBOUT

11OV 60HZ CVT Mainframe uses famous 20 slot GODBOUT Motherboard. 55 lbs.

GBTENC20RM	*20 Slot Rack Mount*	*$895.00*	$825.00
GBTENC20DK	*20 Slot Desk Top*	*$825.00*	$760.00

GODBOUT Mainframe, Less Motherboard & Power Supply - Kit. 23 lbs

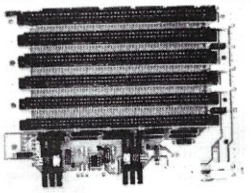
GBTBOX DESK	*Desk Top Main Frame*	$289.00
GBTBOX RACK	*Rack Mount Main Frame*	$329.00

S-100 MAINFRAME - CCS

12-slot motherboard with removable termination card.

CCS2200-01	*Office Cream*	*35 lbs*	*$575.00*	$535.00
CCS2200-02	*Blue*	*35 lbs*	*$575.00*	$535.00

S-100 MOTHERBOARDS



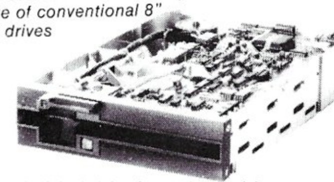
MOTHERBOARD - GODBOUT

Active termination, 6-12-20 slot

GBT153A	*A&T 6 slot, 2 lbs*	*$140.00*	$126.00
GBT153C	*CSC 6 slot, 2 lbs.*	*$190.00*	$175.00
GBT154A	*A&T 12 slot, 2 lbs.*	*$175.00*	$155.00
GBT154C	*CSC 12 slot, 2 lbs.*	*$240.00*	$220.00
GBT155A	*A&T 20 slot, 4 lbs.*	*$265.00*	$235.00
GBT155C	*CSC 20 slot, 4 lbs.*	*$340.00*	$310.00

FLOPPY DISC DRIVES

Tandon TM-800 Thinline is exactly half the size of conventional 8" floppy disk drives



Exactly one-half the height of any other model.
Propietary, high-resolution, read-write heads patented by Tandon
D.C. only operation - no A.C. required
Industry standard interface.
Three millisecond track-to-track access time 9 lbs.

TNDTM8481	*Single Sided* **$495.00** *2 or more*	**$470.00**
TNDTM8482	*Double Sided* **$625.00** *2 or more*	**$600.00**
TNDTM8M	*Manual not included with drive*	**$10.00**

80IR - SHUGART

Single sided doubledensity most popular 8" drive

SHU80IR	*$425.00 ea. or 2 or more (16 lbs.)*	**$395.00**
SHUSA80IRM	*Manual for 80IR drives*	**$10.00**

DT-8 - QUME

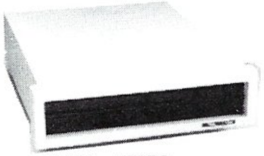
Data track 8 double sided, double density 8"

QMEDT8	*$575.00 ea or 2 or more (16 lbs.)*	**$540.00**
QMEDT8M	*Manual for DT-8*	**$10.00**

5¼" DRIVES - TANDON

TNDTM1001	*Single Sided, 250KB (5 lbs)*	**$310.00**
TNDTM1002	*Double Sided; 500KB*	**$370.00**
TNDTM1003	*Single Sided, 500KB*	**$375.00**
TNDTM1004	*Double Sided, 1000KB*	**$495.00**
TNDTM5M	*Manual, not included with drive*	**$10.00**

DISK CABINETS



V-100 - VISTA

- *Desk or rack mountable*
- *Internal power and data cables*
- *Drives pull out for easy service and maintenance*

VISV100 *Disk Drive Cabinet (35 lbs)*	**$495.00**	**$449.00**

SINGLE 8" - Q.T.

Single 8" cabinet with power supply

QTCDDC8 *(2 lbs)* **$195.00**

DUAL 8" - Q.T.

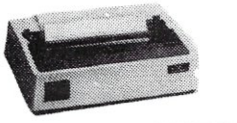
Dual 8" cabinet with power supply

QTCDDC88 *(25 lbs)* **$349.00**

5" CABINETS - VISTA

VIS-9801	*Single 5" with P.S.*	**$75.00**
VIS-9802	*Dual 5" with P.S.*	**$95.00**

PRINTERS



MICROLINE — OKIDATA
WITH FRICTION AND TRACTOR FEED

- *BI-DIRECTIONAL - 120 CPS*
- *9x9 Matrix (Alphanumeric)*
- *6x9 or 12 Matrix for Graphics*
- *5,8.3,10,16 Charactrs p/Inch*
- *6 or 8 Lines per Inch*
- *80CPL@10 CPI for 82A*
- *132CPL@10 CPI for 83A*
- *Parallel and Serial I/O*
- *100 Through 1200 Baud*
- *Self Test*
- *Out of Paper Switch*
- *Friction or Tractor Feed*
- *3"to 14" Top of Form (Switch Selectable)*
- *10 Different Character Sets*

Part No.	Description	List Price	*Sale Price*
OKIDAT82AT	*80CPL @ 10 CPI*	*$ 799.00*	**$539.00**
OKIDAT83AT	132 CPL @ *10 CPI*	*$1195.00*	**$750.00**

MX80 — EPSON
NEED WE SAY MORE?

EPNMX80 *Tractor Feed* *$645.00* **$450.00**

PRINTER INTERFACES - MICROBYTE

RS232 Serial Conversion for MX80

MBSSEI1 *A & T* **$55.00**

Apple Centronics 8 bit parallel interface for Centronics, Epson & OKIDATA printers

MBSAEI1	*A & T*	**$55.00**
MBSAEC1	*Cable for above*	**$14.95**

VISA master charge

PRIORITY ONE ELECTRONICS

MEMBER MITA

9161-S DEERING AVE ● CHATSWORTH, CA 91311

ORDER TOLL FREE (800) 423-5922 CA, AK, HI CALL (213) 709-5464

Terms U.S. VISA, MC, BAC, Check Money Order, U.S. Funds Only. CA residents add 6% Sales Tax. MINIMUM PREPAID ORDER $15.00. Include MINIMUM SHIPPING & HANDLING of $2.50 for the first 3 lbs. plus 25¢ for each additional pound. Orders over 50 lbs. sent freight collect. Just in case, please include your phone no. Prices subject to change without notice. We wil do our best to maintain prices through February, 1982. Credit Card orders will be charged appropriate freight. See November **BYTE** for 60 page Catalog or send $1.00 for your copy today. Sale prices are for prepaid orders only.

NEDA

ples: Integrand, QT). Combined keyboard/monitor/mainframe *not* desirable because of potential maintenance problems. All system components should be individual boards with *rapid* swap of equivalent rebuilt unit via UPS Blue at nominal price as standard service procedure. (Have you ever heard of a high school with a competent inhouse electronics maintenance staff?)

- *Z80, 8080, or 8085 CPU with ROM monitor*. (Future expansion to 8086/8088 if and when adequate software becomes available at reasonable price.)
- *64K dynamic memory*. Why not, 4116's are dirt cheap and the extra reliability of static is not needed for most teaching applications. A unit with less memory should not even be offered. Bank select or extended
- *Good I/O facilities* including at least one, preferably two, RS-232 ports (not merely serial ports) and Centronics printer port (not just unconfigured parallel port), plus keyboard port. At least one plain parallel I/O port or IEEE-488 port for experimental science applications. MP/M support needed as an option.

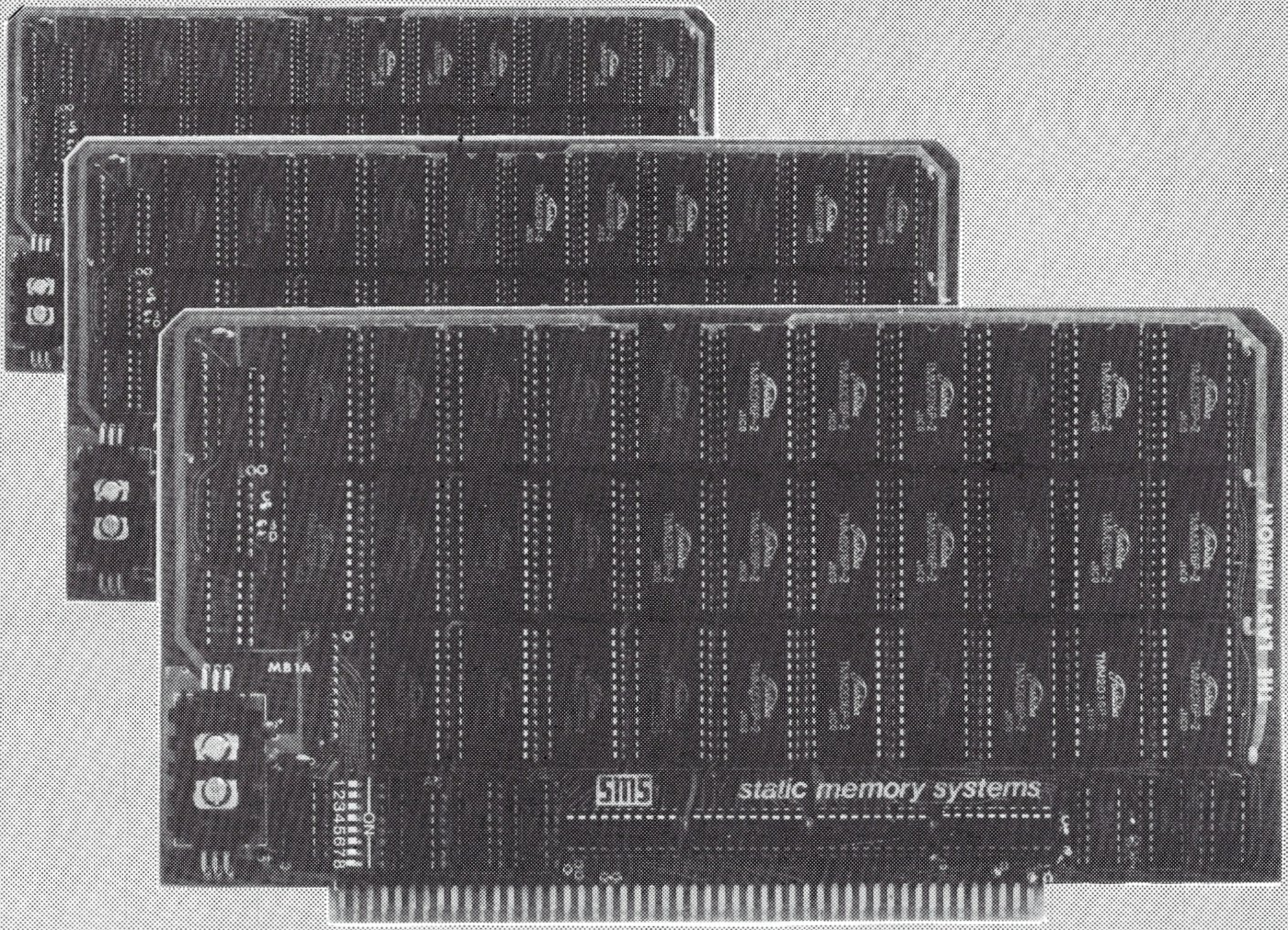
addressing should be available for multi-user systems or extensive college-level applications. The memory should work reliably *without* refresh signal on the bus, to facilitate future expansion to 8086 or 8088.

- *On-board 24 x 80 video, upper and lower case, with good graphics* (say, 160 x 240 or so). Graphics are extremely important for educational work, and S-100 hardware suffers from lack of any video standard, but especially from lack of a graphics standard. Memory-mapped is better than I/O-mapped because graphics are easier (just POKE from Basic) but the board should not permanently occupy main memory space. SSM VB3 would be nearly perfect if it were not so expensive. It costs as much as an entire TRS-80 Model I did! A separate video terminal is *not* acceptable because graphics terminals cost too much. A good monitor (broadband, P-31 phosphor) and keyboard (George Risk is fine) should come with the package. Wiring to the keyboard should be rugged to the point of over-kill.
- *Dual 8-inch disk drives*, both for capacity and because there is no standard for interchangeability in 5-inch drives. No educator will ever have the money to buy all the course application software he needs, or the time to write it. It is urgent that the exchange of software be convenient, and this means 8-inch disk. (The small disks are not cost effective, anyway. If density is held constant, you get three times the storage for 1/3 more money.)
- *Included CP/M with BIOS fully configured* to support all system features. I/O BYTE should be implemented, especially if Microsoft Basic is used.
- *The price for all of this should not exceed $3200*, including CP/M and an inexpensive Basic (TDL/CDL would be fine, although Microsoft would have sales appeal). I personally like Tarbell Basic, but it is a bit hard to work with. Having configured five CP/M systems, I *know* that the stated price is realistic. Even including a good printer (MX80 with graphics, for example), it should be possible to stay below $4000. (I realize the suggested price is somewhat below list for most S-100 vendors. This could be solved via "Educational Discounts" or several other ways. The S-100 manufacturers will soon have to come in at a price below the IBM personal computer, anyway, to survive in any market. To someone who does not understand the advantages of the S-100 bus, the only advantage which this system would have would be 8-inch disks, and this is partly offset by the fact that the IBM unit uses the 8088.)
- *A reasonably priced multi-user version* should also be available. There should be many ways to beat the price of an equal number of PET, Apple, or TRS-80 units. Pricing here is important, but not a urgent as the single-user system. Unless educators can get their hands on a reasonably priced single-user system to try it out, they will never get to the point of buying a multi-user system.

If you have any comments on any of these suggestions, I would be delighted to hear them. I am not sure what I could do, as one individual college professor, but I would be happy to cooperate in any way feasible to bring these ideas to fruition. I believe action is urgent: if you lose the schools, you have lost the future.

Robert J. Hanrahan
Professor of Chemistry
University of Florida
Gainesville, FL 32611

THE LAST MEMORY™



OFFERS MORE FOR LESS

THE LAST MEMORY™, 64K static RAM/EPROM board, sets the industry standard in cost and performance. That's why it's the choice of system integrators, research laboratories, small businesses, large corporations, universities, and hobbyists from Dayton to Tasmania.

Now, how could we make the standard in S100 memory boards better? **BY LOWERING THE PRICE!!**

	KIT	**ASSEMBLED & TESTED**
Board Without RAM	99.99	139.99
16K	219.99	259.99
64K	549.99	589.99

All boards supplied with 150ns RAM.
Any board configuration (i.e. 2K, 4K, ...62K, etc.) available.
OEM discounts available.
All prices are F.O.B. Freeport, IL.
Dealer inquiries invited.

sms ***static memory systems*** *Inc.*

15 So. Van Buren Ave. Suite 209
Freeport, Illinois 61032 (815) 235-8713

MasterCard

VISA®

Letters, continued...

The S-100 UNIX Race Is On!

Dear Editor:

It's always nice to read glowing reports about wonderful new products, and then go out and buy them, take them home and plug them in and have them work as described.

I had this experience recently. I bought the AMCALL package reviewed in *Microsystems*, configured for the PMMI modem card, and it does what is claimed to do, very nicely. Thank you.

It is not so nice to read the same kind of glowing reports, and then, when trying to buy the item, find out that one will probably never be able to buy the it.

I had this experience recently, too. I was impressed by the claims made here and in that *other* magazine about the XENIX operating system. Since I use UNIX at work, I would love to be able to run it at home. Having an 8086 CPU and lots of memory, I ought to be able to run it. Right? Wrong. Sorry, Microsoft says but XENIX can't be run without memory protection—i.e., my brand new IEEE/S-100 memory cards that replaced my pre-standard cards are not good enough?

So how do I go about configuring XENIX for my home system? "You don't. We're only dealing with OEM's" (Original Equipment Manufacturers—a general term for wholesalers). And, the OEM's they have selected are Altos, Codata and others, most of whom are selling hardware in the $12,000-$15,000 price range and who, for the most part, couldn't sell their hardware. The marketing person at Altos didn't even know that the software for the new 8086 system was XENIX.

"Aha!" you say. "Lifeboat has XENIX." Great. I phoned them for details and learned that they have XENIX available, for $2000. It is available off-the-shelf. However, it is configured for the DEC (Digital Equipment Corporation) PDP-11/23 only. This is a very good computer in its field, but it is not a home computer unless you work for DEC and get staff discounts on hardware.

I became very discouraged at this point, and was about to throw in the towel on XENIX. But I dug a little deeper and found not one, but *two* vendors preparing S-100 UNIX versions.

One of these is none other than Lifeboat. They have organized a 16-bit group, and are preparing an S-100 implementation of Xenix. I wouldn't expect anything from this source until the spring of 1982.

The second source is Dual Systems, who have been advertising in *Microsystems* almost from its inception. In the November/December issue, opposite the "Editor's Page" in which you (the editor) stated that ERG was first off the line with an S-100 68000 CPU board, appeared Dual Systems' ad for their 68000 S-100/IEEE-696 CPU board! This board, with its add-on memory management feature, is rumored to be the basis of Dual Systems' S-100 UNIX offering. This will most likely be V7 UNIX (Berkeley UNIX, not Microsoft XENIX), and will probably use either the Konan SMD disk or Dual's own disk controller. They can ship the CPU board now; you might expect to see the software sometime in January.

Whichever of these two groups "gets it together" on time has a shot at getting a good corner of the S-100 UNIX/XENIX market. Best of luck to Lifeboat, DUAL, and anyone else who wants to get into the contest.

Ian F. Darwin
Toronto, Ontario

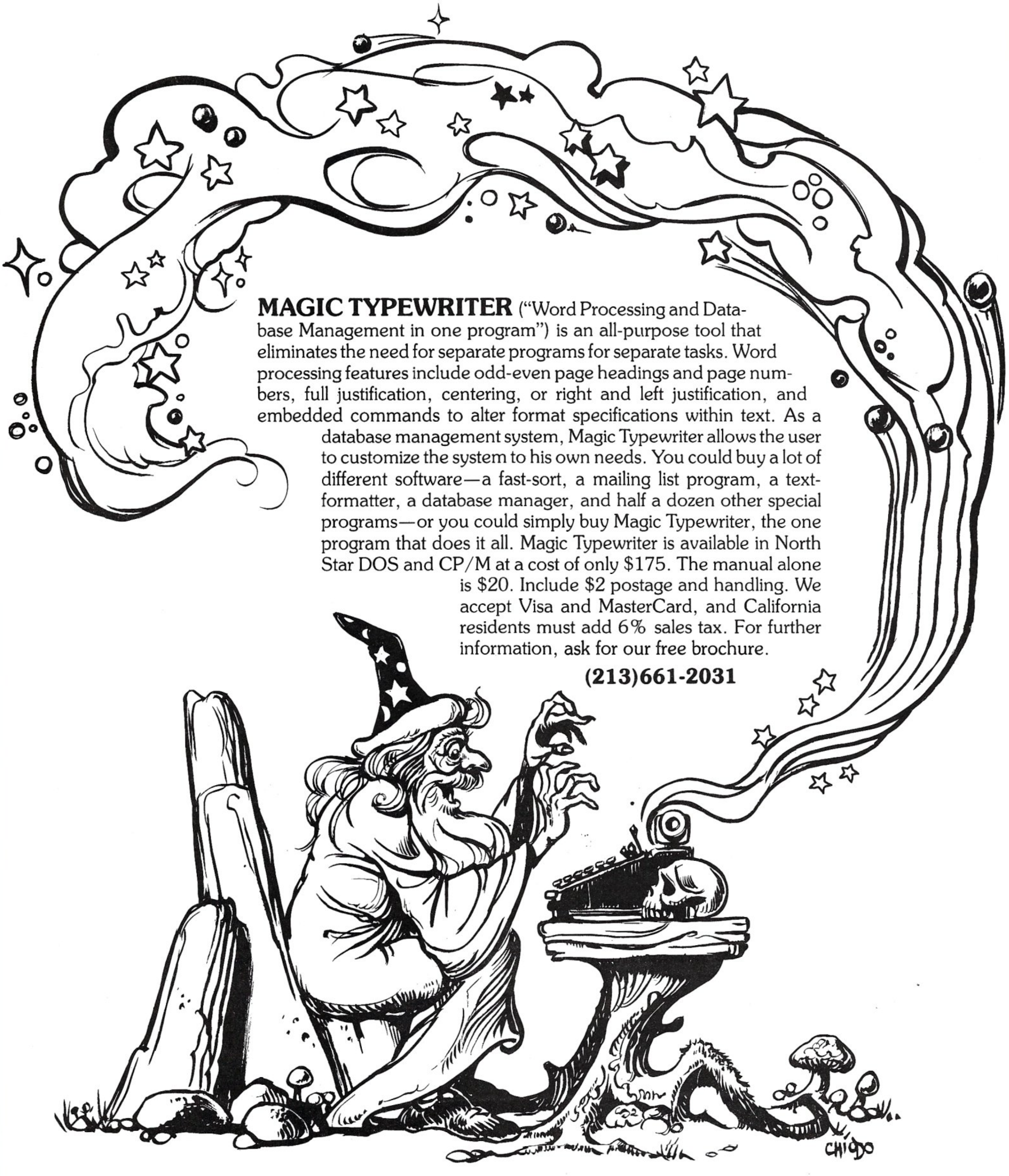
Support for the Versafloppy

Dear Editor:

As a follow up to the letters from Ivan Berger and Robert Luckley, your readers may like to know that FBN Software can provide support for the Versafloppy disk controller. Our address is 1111 Sawmill Gulch Road, Pebble Beach, CA 93953.

We have a BIOS for both versions of CP/M as well as a number of utilities for different disk formats. This software is available for the cost of the media and a nominal handling charge.

Nick Hammond
FBN Software
Pebble Beach, CA



MAGIC TYPEWRITER ("Word Processing and Database Management in one program") is an all-purpose tool that eliminates the need for separate programs for separate tasks. Word processing features include odd-even page headings and page numbers, full justification, centering, or right and left justification, and embedded commands to alter format specifications within text. As a database management system, Magic Typewriter allows the user to customize the system to his own needs. You could buy a lot of different software—a fast-sort, a mailing list program, a text-formatter, a database manager, and half a dozen other special programs—or you could simply buy Magic Typewriter, the one program that does it all. Magic Typewriter is available in North Star DOS and CP/M at a cost of only $175. The manual alone is $20. Include $2 postage and handling. We accept Visa and MasterCard, and California residents must add 6% sales tax. For further information, ask for our free brochure.

(213)661-2031

CDE CALIFORNIA DIGITAL ENGINEERING
P.O. BOX 526 ★ HOLLYWOOD, CA 90028

The CP/M* Bus

by Anthony Skjellum

Last issue, we discussed the concept of the sub-directory as a possible enhancement to CP/M. This month we will complete the discussion of possible enhancements for CP/M2 with the description of the internals of link records discussed in the July-August column.

There are two additional items this month—a description of ZDM, a Z80 debugger, and a discussion of register usage in CP/M BDOS calls.

More Features for CP/M Part IV

Last issue, we introduced the concept of the link file. This type of file structure was designed to allow sophisticated data manipulation by permitting files to reference data belonging to other files. Two types of link files were proposed: the simple link file and the complex link file. Simple link files provided a limited but useful way to eliminate the need for multiple copies of the same file on a given disk. On the other hand, complex link files provided a very general way to create data structures. We will now discuss the internal format of link records used in complex link files.

As stated before, link records consist of information which tells the BDOS what file or part of a file to access. First, the record itself is initiated by a ^Y character and is the same length as the file record length. This marker serves to distinguish it from a data record. For standard files this is 128 bytes; for VLR files it must be at least 16 bytes. In the July-August column, we did not clearly distinguish between link records and their contents (link instructions). We stated that the "maximum length of a link record is 16 characters..." What this means is that the maximum length of one line instruction in a link record is 16 bytes. This is the smallest meaningful complex link request which can stand alone, which is why we require VLR files to have at least 16 byte records if they are to use complex links. Furthermore, it should be noted that files with sufficiently large record length are permitted to have multiple link instructions per line record.

Now let's illustrate the format of link records and their instruction sequences. First, a standard instruction:

```
     ^Y        ; indicator.  Also is the first
               ;  character of the first link
               ; instruction for this record

               ; Bit seven of the first character
               ; is used internally to indicate
               ; if this is the last instruction.
               ; It is set high for the last
               ; instruction in this record.

FILENAME.EXT   ; 11 characters long
```

```
<start>   ; 16 bit quantity indicatingthe
          ; first record that BDOS should
          ; read.
<end>     ; 16 bit quantity indicating the
          ; last record BDOS should read.
          ; If <end> is zero, the file is
          ; read to eof.
```

We can abbreviate this as ˆ Y,FILENAME.EXT,<start>, <end> and will do so with further examples for the sake of brevity. We will write ˆ Y! when the last command indicator is set.

So far we have defined the basic command structure for a link instruction. Imagine the case in which we will link to several portions of the same file by contiguous link instructions. In this case, it becomes convenient not to require that the filè be named again. (In fact, it is preferable not to re-open the file either for the sake of efficiency.) This is done by creating a command character in addition to ˆ Y. We now add the character ˆ X as a second link indicator. A ˆ X tells the BDOS that the file to be read was just read from and that only the <start> and <end> specifications follow. This link instruction is therefore only five bytes !ong. Once again, if this is the last instruction for the current record, the ˆ X is replaced by ˆ X! (i.e. ˆ X+80H).

We will illustrate this concept with the following example. We want to have a file link to three portions of a file named DATA.TXT. This will all be done in one link record by way of three link instructions. Again, we assume the standard CP/M record length of 128 bytes:

```
^Y,DATA.TXT,1,5    ; read sectors 1-5 inclusive
^X,7,9             ; read sectors 7-9 inclusive
^X!,20,0           ; read sectors 20-eof inclusive
                   ; and mark end of instructions.
```

This would result in the following code in the link record (represented in hexadecimal):

```
19 44 41 54 41 20 20 20 20 54 58 54 01 00 05 00
^Y D  A  T  A  bl bl bl bl T  X  T   0001  0005
18 07 00 09 00
^X  0007  0009
98 14 00 00 00
^X! 0014  0000
```

(with random bytes to end of record which are ignored) Remember, if DATA.TXT contains any link records in the portions read, they too will be returned as part of the operation. As discussed in the July-August installment, the extended file control block (EFCB) facilitates this; it must also be large enough to support the nesting depth actually used in the files.

Several points are noteworthy. First, if indexing schemes are created with complex links, the ˆ X mode will be avoided to simplify sorting of the link records. Also, note that ˆ X sequences may not be continued past the end of a link record (same length as data records: 128 bytes for standard CP/M file). A ˆ Y sequence must begin each new link record since it also serves to identify the record as such. This is not a serious restriction, as link records may be packed as desired. Also, a single 128 byte record will contain 22 ˆ X sequences following the initial ˆ Y request. This would be quite sufficient for most applications. Of course, VLR files which use record lengths greater than 128 bytes may also have correspondingly longer link expressions.

Having explained link records and link instructions, we have completed the discussion of the complex linking

NEW LOWER RAM PRICES

LDP1/2

The LDP1/2 utilizes the advanced 8088 processor to provide up to 8 times the throughput of a 4 MHz Z80A processor. The powerful instruction set of the 8088 is ideally suited to higher level languages such as PASCAL and PL/1. The 10 slot motherboard leaves 7 slots for USER expansion. With the option of a 10 MByte Winchester and MP/M-86, the LDP mainframe becomes a powerful multiuser system with the capability of handling 8 users without the degradation in performance experienced with Z80 CP/M systems. The performance of the LDP1 and LDP2 has never been available for such an affordable price.

FEATURES:
- LDP88, 8088 CPU board
- LDP72, advanced floppy disk controller
- LDP128K dynamic RAM
- 1 serial RS232 port
- 10 slot mainframe
- 18" floppy drive (LDP1), 2 8" floppy drive (LDP2)
- 4K EPROM socket for user population
- ROM resident diagnostics
- CP/M-86

OPTIONS:
- HAZITALL
- 8" 10 MByte Winchester
- MP/M-86 multiuser system
- Woodgrained 7 slot chasis

PRICES:

LDP/1 w/128K RAM	$3295
LDP/2 w/128K RAM	3895
LDP/2 Winchester, 128K RAM	6795

SOFTWARE:

CP/M-86	$250
BASIC	450
FORTRAN	450
FORTH	100
C	150

LDP88 8088 CPU BOARD

• 8088 CPU 5 MHz operation upgradeable to 8 MHz • 9 vectored interrupts • Fully complies with IEEE 696 electrical and timing specs • RS232 serial port with modem controls • 1K bytes of static RAM • 2 EPROM sockets (2716 or 2732) • 8087 upgrade kit available in Sept. • 8 bit bus eases interface to other S100 bus boards • 1MByte address space • 65K I/O ports

PRICES: Kit $319.95
Assembled & tested $349.95

64/256K MEMORY

• 8 or 16 bit operation • Meets all IEEE 696 specs • Access time 350 ns from PSYNC low • Intel 8203 dynamic RAM controller • 24 or 16 bit address decoding • No wait states with 5 MHz 8088 or 8086 • Parity with Error interrupt generation • No DMA RESTRICTIONS • 64K board is upgradeable to 256KByte board

PRICES:		
	64K	$595
	128K	795
	256K	1395

HAZITALL

• 2 Serial RS232 ports • 2 parallel ports with handshake control • Math processor support (8231/9511 or 8232/9512) • WINCHESTER DISK support • Real time programmable interrupt • Clock/calendar with battery back up • Synchronous data communication supported

PRICES: Kit $249.95
Assembled & tested $325.00

LDP72 FLOPPY DISK CONTROLLER

• IBM compatible single and double density format • Single or double sided drives • Programmable data record length (128 to 8192 bytes/sector) • Multi sector and multi track transfer capability • Parallel seeks on up to 4 drives • On board digital data separator • Software selectable single or double density operation • Separate connectors for 5 1/4" and 8" drives • Software selection of standard or minidrives allowing mixing of both drives on a single controller

PRICES: Kit $219.95
Assembled & tested $274.95

Watch this ad for the announcement of the most powerful S-100 Bus CPU board available.

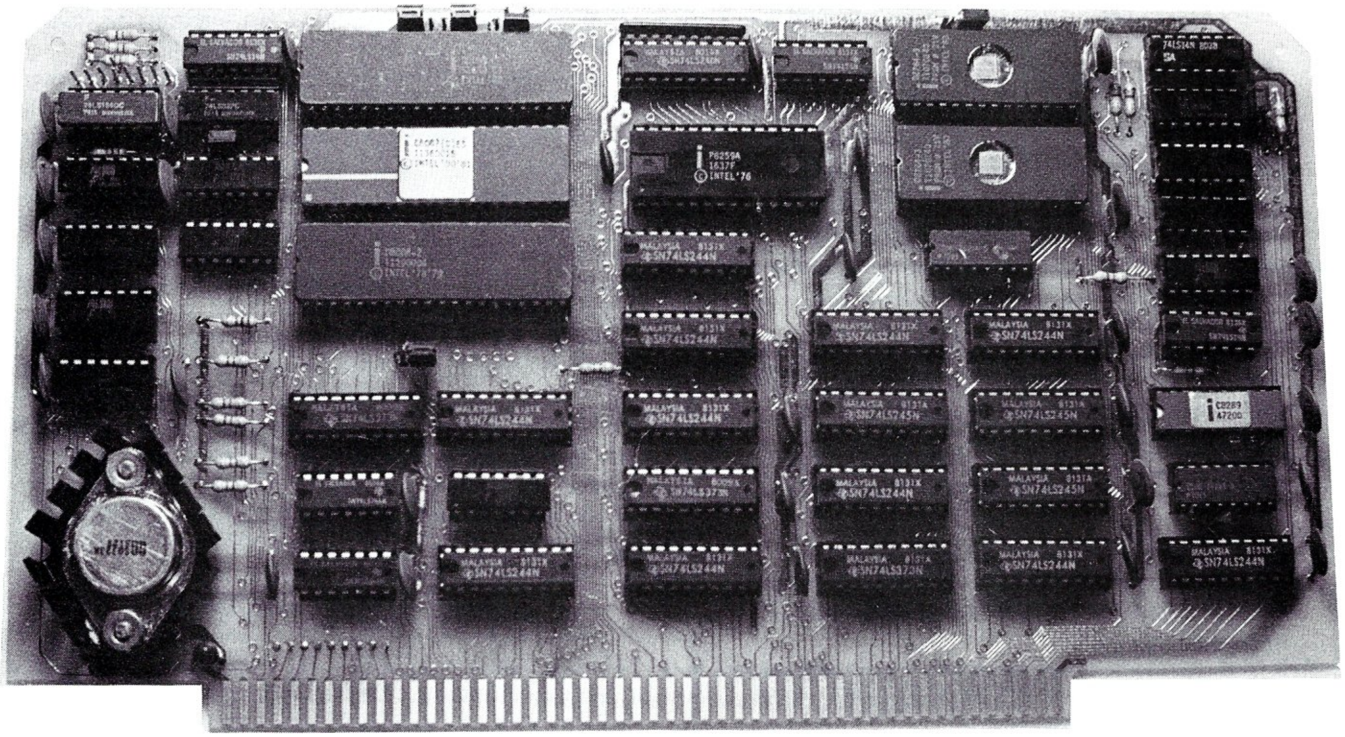
Lomas Data Products

11 Cross Street
Westborough, MA 01581
Telephone (617) 366-4335

Introducing the fastest processor board available on the S100 bus today...

The Lightning One™

8086/8087/8089 CPU Board



Features:

□ 4, 5, 8 or 10 MHz operation jumper selectable
□ 8086 or 8088 main processor
□ Independent I/O and memory waitstate generator
□ Up to 16K bytes of onboard EPROM
□ 9 vectored interrupts expandable to 65
□ EPROM monitor with diagnostics and disk utilities
□ 8087 and 8089 co-processors available onboard
□ CP/M-86* and MS-DOS** software support

The *Lightning One*™ contains not one processor, but three processors all working in parallel. It uses the Intel 8086 as the main processor. The math capability is augmented with the 8087 math processor and the I/O handling capabilities are augmented with the 8089 dual channel I/O processor. The board complies with all IEEE-696 specifications.

If you have an 8 bit system presently, don't despair. The *Lightning One* is available with the 8088. The 8088 is fully software compatible with the 8086, but utilizes an 8 bit bus allowing use of your present 8 bit memories. When you are ready to upgrade to full 16 bit operation, you need only to unplug the 8088 and plug in an 8086 in its place. When using an 8088, the 8087 and 8089 may still be utilized.

In addition to the *Lightning One,* Lomas Data Products has a full line of S100 bus support cards including: memory, disk controllers, and serial and parallel I/O.

Prices for the Lightning One start at $425.

For 16 bit computing on the S100 bus, come to the leader . . .

LOMAS DATA PRODUCTS

11 Cross Street
Westborough, Massachusetts 01581
Telephone: 617-366-4335

**CP/M-86 is a trademark of Digital Research.*
***MS-DOS is a trademark of Microsoft.*
Lightning One is a trademark of Lomas Data Products.

CP/M Bus, continued...

process. Such an added feature could be put to very good use in developing sophisticated systems software and application programs under CP/M.

ZDM: Z80 Debugger

ZDM is a noteworthy software package. It is a useful Z80 debugging tool with good documentation. ZDM is $45.00 and is marketed by RD Software of Pacific Palisades, CA. Styled after the DDT transient, it supports Zilog and TDL-like mnenonics. I discuss ZDM because it is an example of inexpensive software of high quality.

Documentation is a crucial part of a software package. Many discount packages provide only scanty documentation a serious drawback. This is not true of ZDM. ZDM's documentation is concise: it says everything that needs to be said. DDT users will have no problem adjusting to ZDM and backup instructions are included to help the novice.

Software should work as described. Inexpensive software is usually quite limited in capability and sometimes has serious limitations which become obvious only with use. Not ZDM: it works as advertised.

Software should also be able to accomodate your display system. For example, DDT won't work properly with 64 character displays, while ZDM can be initialized for the proper printing width and number of lines to print for its 'L' and 'D' commands.

My point is that well-written software need not cost so much. ZDM provides a full-featured debugger and includes the information needed to use it effectively. It is not limited like other inexpensive software, but has the power necessary to provide a fine debugging environment. Users should think seriously about supporting vendors who produce software of the same caliber and price range as ZDM. (For more information on ZDM, see the review in *Dr. Dobb's Journal*, February 1981.)

Register Usage in BDOS Calls

This section is in response to a reader inquiry concerning register usage in BDOS calls. First, note that all 8080 registers are modified during calls to the location BOOT+5H. This requires that any program calling BOOT+5 take proper action to save any registers beforehand whose values must be used after the call. Since CP/M was written for the 8080, it is safe to say that the additional Z80 registers (IX, IY and the alternate 8080 set) will not be changed within BDOS. However, the BDOS calls the system BIOS routines which are customized for each CP/M installation. These subroutines could utilize these other registers, so considerable caution must be exercised before assuming that CP/M will not modify a given register. Furthermore, software which assumes things about a given BIOS will not be portable. Therefore, it is advisable to assume the worst case so that the program will not be tied to a single machine. ■

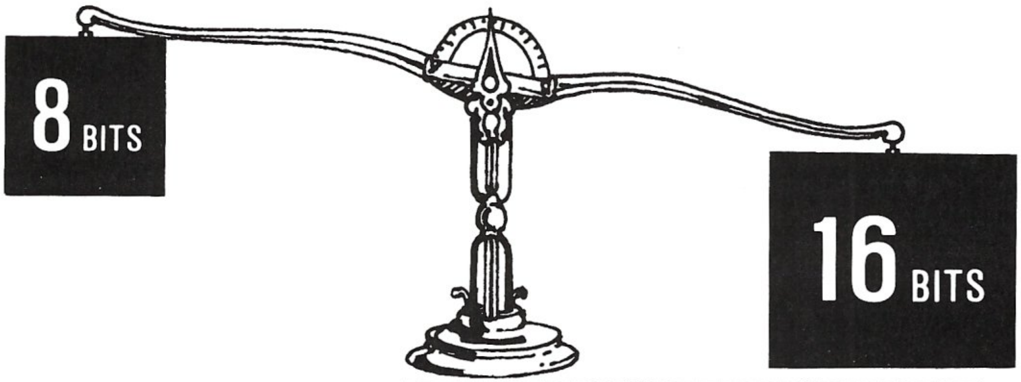
Errors In The Last Column

We regret two errors that occurred in the last "CP/M Bus." They were the following:

1) The first two examples shown were positioned in reverse order.

2) In third from last line of the text, the phrase "terminated by Z" should have read "terminated by ^Z."

The editors wish to apologize to the author and our readers.



MICROCOMPUTERS

THE CHOICE JUST GOT EASIER!

PRICE $4390 Gets you a Tec-86 Complete 16 Bit 8086 System, including 64K Bytes of Memory, Dual 8 Inch Double Density Floppy Disks, and more. . . (See complete Tec-86 specs below.)

PERFORMANCE Faster Processing, Increased Address Space, up to One Megabyte of available memory, and the Increased throughput of the the TEC-86 16 Bit System means that you can now do work on a microcomputer never before practical with the slower 8 Bit systems.

SOFTWARE CP/M-86™ Brings the World of CP/M software up to the 16 bit systems. BASIC-86™, MS-Fortran™, MS-PASCAL™ are just a few of the Languages currently available.

EXPANSION Memory Expansion up to ONE MEGABYTE. Disk Storage can be increased to over 30 Megabytes with our Winchester Disk Subsystem. Multi User Systems available soon!

QUALITY Full ONE YEAR WARRANTY is standard on TECMAR components. This Warranty is possible due to our High Engineering Standards, and Thorough Product Testing.

DELIVERY We have been delivering 16 Bit Boards and Systems for nearly two years. Most orders are shipped from stock or within 30 days.

S100 A/D $765
Analog to Digital Converter with Timer/Counter

12 Bit Accuracy, 16 Channels,
30 KHz Conversion, Five
Programmable Timer/Counters for:
- Time of Day
- Triggering Conversions
- Counting Conversions
- Counting External Signals
- Interrupting the CPU

Options: Programmable Gain;
14, 16 Bit Accuracy;
40, 100, 125 KHz Conversion;
Expansion to 256 Channels

$4390 TEC-86 $4390
16 BIT 8086 MICROCOMPUTER SYSTEM

FEATURES: 5 MHz 8086 CPU * 64K Bytes RAM * S100 IEEE-696 * Dual 8 Inch Double Density Floppy Disk Drives * Two RS 232 Serial Ports, 50 to 19200 Baud * Three Parallel Ports * ROM Boot for CP/M-86 * Heavy Duty Power Supply * Attractive Industrial Quality Desk Top Cabinet * Completely Assembled and Tested * **OPTIONS:** Up to One Megabyte of Memory, Double Sided Double Density Floppy Disks * Up to 31 Megabytes of Winchester Fixed Disk Storage. * CP/M-86 * BASIC-86 * MS-FORTRAN * MS-PASCAL * CBASIC/86™ * PASCAL/M86™

$10990 TEC-86W $10990

Same Great Features as our TEC-86 except that the TEC-86W has 256K of RAM, and our 31 Megabyte Winchester Disk Subsystem.

NEW — 64K/256K S100 MEMORY $595/$1695

8 and 16 Bit Transfer for 8 and 16 Bit Systems
MP/M and MP/M-86 Compatible,
Up to 7 users on one 256K board.
Parity Detection with Interrupt and LED Indication
Hardware Write Protect
Full DMA Capability
24/20/16 bit Address Decoding
Enable/Disable 8K/32K Segments
Bank Select in 8K/32K Segments
Meets IEEE-696 Specifications

See our catalog for many other fine S-100 and Apple Boards including: — Analog to Digital Converters *(16-256 channels; 12, 14, or 16 bit accuracy; 30, 40, 100, or 125 KHz; programmable gain; timer/counters)* **— Digital to Analog Converters** *(12 bit accuracy, 3 microsecond conversion)* **— 8086 CPU Board — I/O Boards — 16K Static RAM — Real-time Video Digitizer and Display.**

CP/M MP/M CP/M-86 and MP/M-86 are trademarks of Digital Research Inc. BASIC-86, MS FORTRAN and MS-PASCAL are trademarks of MicroSoft Inc. PASCAL/M86 is a trademark of Sorcim CBASIC/86 is a trademark of Compiler Systems Inc.

DEALER INQUIRIES INVITED

23600 Mercantile Rd. • Cleveland, OH 44122

TECMAR, INC.
(216) 464-7410

PL/I For Limited Resource Computers

by Gary Kildall

PL/I, Programming Language One, has in one form or another been with us for nearly twenty years. Although a pragmatic language, it was considered large, unwieldy, and difficult to implement. Recently, however, the language has been revitalized through the efforts of the American National Standards Technical Committee X3J1 (1) where the General Purpose Subset language was defined. This so-called "Subset-G" language is upward compatible with full PL/I, but is designed expressly for minicomputer implementation. The elements selected for inclusion within Subset-G are the most commonly used facilities used in commercial, scientific, and educational application programming. Redundant language constructs, little-used facilities, and error-prone statement forms were eliminated, resulting in a sub-language which most observers believe is superior to the full language in many ways.

ANSI Standard Subset-G is now available for operation on several minicomputer systems, including the Data General Eclipse and MV/8000, Prime Computers and the popular Digital Equipment Corporation VAX computer. PL/I-80, offered by Digital Research, is based upon Subset-G, and brings many minicomputer and mainframe facilities to the microcomputer application programmer. The following is a brief history of the PL/I language, a discussion of PL/I facilities, and an overview of the Digital Research implementation.

PL/I was originally conceived in the early 1960's by the Advanced Language Development Committee of the Share Fortran Project in the wake of interest created by Algol, Fortran, and Cobol. Elements of each of these languages were incorporated into the original design: block structure, nested scope of variables, procedure formats, and array refenencing were, like Pascal, derived from Algol. Scientific facilities came from Fortran, including separate compilation expression formulation, floating point arithmetic, some I/O formation, and a wide variety of transcendental functions. Commercial processing in PL/I was derived from Cobol, including structures, decimal arithmetic, file processing, and picture formats. A variety of new statement forms were added to allow character string processing and error-exception handling, which were considered essential for high-level application programming. Real-time multi-tasking facilities were also added to allow PL/I to be used for systems programming as well. The language which resulted from this design effort contains more built-in data types, arithmetic operations, and general-purpose programming facilities than any other programming language available today. But herein lies the primary difficulty with full PL/I. The language is too large to implement effectively on any but the largest mainframes. The complexity of the language also inhibited proper use of all language features, while the unwary programmer was often trapped by strange twists and nuances of the language. Nevertheless, PL/I has proved to be a practical, pragmatic language for application programmers over the past several years, through implementations on a variety of mainframe computers.

Gary A. Kildall, Digital Research, Inc., Pacific Grove, CA.

Reprinted from NCC 1981 Proceedings with permission of the author. PL/I-80, MP/M, CP/NET and CP/M are registered trademarks of Digital Research.

The popularity of PL/I led to standardization efforts for the full PL/I language. The document produced by the ANSI committee for full PL/I gives complete syntactic and semantic specifications for the language in a form suitable for compiler and run-time system implementation. That is, the language specification describes the manner in which PL/I must be implemented in order to conform to the standard, but does not specifically cover PL/I programming practices. The full PL/I document is considered one of the best language specifications produced to date.

The Subset-G document, in turn, describes the portions of full PL/I which are to be included. Specific features which remain in Subset-G include:

Decimal arithmetic
Character and String Constants
Restricted Array and Structure Assignments
Allocate and Free
Record (binary) I/O
Stream (ASCII) I/O
Format Specifications with Pictures
On-Conditions
A wide variety of Built-in Functions
Separate Compilation
Initialized Variables
Based Variables

The Digital Research PL/I-80 programming system project was started in 1978, and completed two years later. PL/I-80 is based upon Subset-G, with nearly all of the Subset-G features, and operates under the Digital Research CP/M, multiprogramming MP/M, and CP/Net network operating systems for 8080, 8085, and Z-80 microprocessors. The PL/I-80 programming system itself consists of the compiler, macro assembler, linkage editor, program librarian, and run-time subroutine library.

The PL/I-80 compiler is a "three-pass" system that reads a PL/I source program prepared using a program editor, and produces a relocatable file as output. The first pass collects declaration information, and produces a symbol table used by subsequent passes. The second pass augments the symbol information and produces intermediate language in tree-structure form for subsequent code generation. Both passes analyze the source program using recursive descent.

The third compiler pass is largely machine-independent, and consists of a comprehensive code optimization system, along with semantic handlers for 8-bit code generation. The optimizer processes the intermediate tree structures in three stages: first the trees are "normalized" and "flattened," then analyzed by a "frame optimizer," and finally processed by a "special-forms recognizer."

The normalization and flattening process reduces alternate forms of an equivalent expression to the same form, while rearranging expressions to reduce the number of intermediate temporary variables. The frame optimizer performs common subexpression detection within a limited range of tree-structures in preparation for later processing. This limited window provides optimizing information over a range of approximately ten to twenty statements, thus avoiding the processing overhead associated with complete program flow analysis. Trees annotated with optimizing information are then passed to the special-forms processor where approximately three hundred tree-structures of special interest are matched and detected. Special-forms recognition allows concise sequences of code to be produced for many common statements.

As an example, suppose the statements shown below occur in a PL/I-80 source program:

```
K=1 + J
I=J + I
A(I)= A(K) + I
```

The normalization process rearranges the first statement to

```
K= J + I
```

The frame optimizer then marks I and K as equivalent expressions so that A(I) and A(K) are known to have the same address. The special-forms recognizer notes that the A(I) array element is simply being incremented, and thus produces an increment memory instruction to affect the operation.

Generally, the PL/I-80 optimizing scheme produces dense machine code for all operations which are reflected in 8-bit and 16-bit architectures, including byte and word fixed point and bit string operations. More complicated data forms, such as floating point and decimal arithmetic, are performed out-of-line by calls to subroutines extracted from the run-time library.

The PL/I-80 linkage editor combines relocatable code produced by the compiler and macro assembler into a machine-executable memory image. In addition, subroutines are automatically extracted from the PL/I run-time library when referenced. The linkage editor also allows multilevel overlays so that a large application, such as a menu-driven inventory control program, can be effectively executed in a small memory region.

The PL/I-80 programming system is currently being transported to 16-bit processors, with initial support for the Intel 8088 and 8086 processors so that designers may select either 8-bit or 16-bit host processors for their application programs. The transition to the Intel processors is simplified in two ways. First, the compiler itself is written in PL/M, Intel's high-level system language, with portions of the run-time system written in PL/I. Thus, only the semantic handlers need to be altered, along with conversion of the space and time critical run-time subroutines, such as the floating point library, which are implemented in assembly language.

The PL/I programming system will be transported to all processors and operating systems supported by Digital Research in the future, and serves as the basis for application software written for the microprocessor industry by independent software vendors.

Subset-G is a concise, consistent and practical language for professional programmers who write quality commercial application programs for their own use or for public distribution. Further, the rapid acceptance of the Subset-G standard in the minicomputer industry opens a wide customer base for application programs, while ensuring that those programs will not become obsolete. ■

Reference

1. *ANS Programming Language PL/I General-Purpose Subset* (BSR X3,74), American National Standards Institute, 1430 Broadway, New York, NY 10018.

Programming Style Comparisons: Digital Research PL/I-80 and Microsoft Basic

by Michael J. Karas

I recently purchased a copy of a newly introduced applications programming development system that is a microprocessor—oriented implementation of the PL/I programming language. This language, developed by Digital Research Inc., Pacific Grove, CA, is a powerful, structured compiler that is based upon the ANSII Standard PL/I Subset G. Gary Kildall of Digital Research has made this package compatible with the CP/M operating system for use on 8080/8085/Z80 microprocessors. The machine code developed by the compiler is also compatible with the above family of microprocessors.

PL/I was originally developed by IBM as a large system language that was to be the last answer for programming languages in that it contained capabilities similar to Fortran, Cobol, Algol, and Pascal. Over the years since its introduction, PL/I has gained a reputation and a "following." The reputation has been that the compiler is huge and that the machine code modules produced are also huge (i.e. many bytes of memory are needed to run the compiler and the resulting programs). The "following" is a growing number of systems and applications programmers that have come to know the power of PL/I, ease of developing programs, degree of self-documentation within programs, structure, and ease of maintaining programs. As the language gained in popularity, several manufacturers of minicomputers (including DEC) developed subset implementations of PL/I for their machines. Somewhere along the line a committee was formed to develop an ANSII standard PL/I subset to permit program transportability. Note that the idea of developing a PL/I subset was not to remove power or to limit capability of the language. Instead, much redundancy and feature overlap was removed to make the compiler requirements smaller and manageable in "minicomputer amounts of memory."

About three years ago, Gary Kildall of Digital Research saw that there would be a need and a market place for a good compiler-type language for microprocessors. He decided that PL/I in a subset form would be the way to go. The results of his (and I'm sure also that of others at Digital) efforts is PL/I-80. This compiler, in my opinion, is the best thing to happen to the microprocessor field in several years. For those people who are serious applications programmers trying to develop sophisticated applications packages, the PL/I-80 system is the answer. The reasons are many, some of which were mentioned in the previous paragraphs. In using PL/I-80 for several months now, I have to say that the programs made in PL/I are blessed with the following advantages:

a) Structure is inherent in the programming style.
b) Programs are extremely self-documenting.
c) The compiler is fast and makes efficient code that also runs fast!
d) Linkage of programs to assembly language or the CP/M operating system is easy.
e) The linkage capability includes a simple to use and powerful library and overlay generation capability.
f) Data types included in the language are comprehensive.

The PL/I-80 system has some minor disadvantages that I'll mention just to set things straight for all those people who are hung up on other languages. As it is a compiler, the development time from coding to running code is much longer that an intrepreted language like Basic. The program development time for the inexperienced PL/I programmer will generally be about five times as great as programming in Basic. I have found that, as I gained familiarity with the PL/I-80 features and had the use of a good screen mode video editor, program development time was somewhat less than twice that of Basic.

Michael J. Karas, 2468 Hansen Ct., Simi Valley, CA 93065

Also, small programs seem to turn into very large .COM files. (Small programs that do any I/O get large quickly due to included device and file interface code.) The real efficiency of the generated code is felt when source code programs start to get up into the hundreds of lines. Small programs will typically be 60 to 100 lines, and will generate 9 to 15 kbytes of machine code. Once large portions of the runtime library become utilized by the program's logical contents, then each additional 20 to 30 lines of code may only add a portion of a "Kbyte" to the .COM file.

Program checkout also tends to be harder if you desire to be a "seat-of-the-pants" programmer. Minor editing and logic mistakes cause a lot of time to be consumed in re-editing, re-compiling, and re-linking the program that is under development. Most Basic programmers I know, including myself, tend to design and kludge programs right at the console due to the immediate testing convenience offered by an interpretive language. A more serious programmer will tend to "design and conceptualize" a program ahead of the coding process. This makes for a better, more structured, and logically correct program. (For additional thoughts on design and conceptualization see Greg William's editorial in the March 1981 *Byte*, page 6). When I make a PL/I program it tends to be developed by the latter process. The linking capability also allows programs to be developed in modules so that editing and manipulation of programs is done in small segments.

I often get asked the question, "What is PL/I-80 like?" In an attempt to answer that question the remainder of this article will present a programming problem in both Microsoft Basic and Digital Research PL/I-80. The aim is neither to try to teach the reader how to write PL/I-80 programs nor to demonstrate the full power of the language. I also want to stress that while this example can be implemented in either Basic or PL/I-80, it should be noted that the latter language will generally provide a much better vehicle for implementing complex algorithms into a program, due to the available language features and constructs. Obviously this cannot be demonstrated in a short magazine article. The following examples are meant primarily to show one simple method of solving the same problem in two different computer languages. The comparison to Basic was selected due to the fact that more readers probably know Basic than any other language. This will permit the most enjoyable contrast of programming styles for those readers just now getting their first glimpse of PL/I-80.

The example programs are both designed to perform exactly the same function. The idea is to read a name/address/telephone number file in sequence. The data is then formatted to list the names and addresses in a format compatible with mailing labels on the printer. As each name is printed on its label, the name and telephone number are listed on the console. This example is designed for demonstration purposes only and may not represent an efficient or feasible implementation of the above name and address algorithm.

The input data to the program is expected to exist in a file named "NAMES.DAT" on the default CP/M disk unit. The demonstration format for this file and some test records are shown below. Note that the data may have been formatted into these records by another program or through use of an editor. Also the field length is only typical and may be impractical in a real processing application. Field identification is as follows:

last name, 8 bytes
first name, 8 bytes
middle initial, 1 byte
address, 20 bytes
city, 11 bytes
state, 2 bytes
zip code, 5 bytes
phone number, 10 bytes

Records are terminated with carriage return and line feed pairs.

Example Input Data

```
KARAS   MICHAEL J2468 HANSEN CT          SIMI VALLEYCA930658055277922
SMITH   KELLY   S3055 WACO AVE           SIMI VALLEYCA930638055270518
JOHNSON JACOB   B2793 ANDREW COURT       MANKADO    MN567056123424469
ERICKSONSHIELA  P454 B UNIVERSITY AVEFARGO      ND581024154435523
WILLIAMSDALE    F35912 CIRCLE MTN DR DAVENPORT  IW590342179557451
HANSEN  MICHAEL H2486 KARAS CT           SIMI VALLEYCA930658055279355
KARAS   HANSEN  J8324 MICHAEL CT         SIMI VALLEYCA930658055277922
NIXON   GERALD  R9355 KENNEDY BRIDGE OVER RIVER MS204312225551212
PAULSON DEBBI   Q6699 HOLLOW TREE RD ROLLING LOGIL569433984535551
```

The following program listing presents a PL/I-80 implementation of a program to read the above data and print the console list and the mail labels at the printer. Observation of the program structure and the various PL/I-80 constructs is left for the reader.

A Short Mailing Label Printing Program in PL/I-80

```
maillab:
        procedure options(main);
        dcl                                /* declare all variables */
                database        file,
                syslist         file,    /* printer output name */
                nulstr          char(1) varying,
                1 record,                /* a structure for data */
                  2 lastname    char(8),
                  2 firstname   char(8),
                  2 midinit     char(1),
                  2 address     char(20),
                  2 city        char(11),
                  2 state       char(2),
                  2 zip_code    char(5),
                  2 phone_num   char(10),
                  2 filler      char(2), /* cr-lf filler from record */
                1 phone_format  based(p),/* overlay template for phone #
                  2 area        char(3),
                  2 prefix      char(3),
                  2 line        char(4),
                p               pointer;

        open file(database) input record sequential title('names.dat');
        open file(syslist) output stream print title('$lst');

/* setup what to do if we try to read the file and there is no more data

        on endfile(database) begin;
                                close file(database);
                                put file(syslist) skip;
                                stop;
                             end;

/* tell operator to put mail labels into printer */

        put edit('Put mail labels into printer.',
                 'Type "GO" when ready....')
                (skip(3),a,skip,a);
        get list(nulstr);

/* set address pointer for phone format overlay template */

        p=addr(record.phone_num);

/* read data into structure and then print the mail labels */

        do while('1'b);             /* a forever loop!? */

                read file(database) into(record);

                /* put names and phone numbers to console */

                put skip edit(strip(firstname),' ',
                              strip(lastname),' (',
                              area,') ',prefix,'-',line)
                          (3(a),col(30),6(a));

                /* print names to printer on mail labels */

                put file(syslist) edit(strip(firstname),midinit,
                                  '.',strip(lastname),
                                  strip(address),strip(city),
                                  state,zip_code)
                             (skip(3),a,x(1),a,a,x(1),a,skip,a,
                              skip,3(a,x(2)));
        end;

/* function routine to return a string of varying length based upon
   input string with all trailing blanks removed */

strip:
        proc(string) returns(char(50) varying);
        dcl
                string  char(50) varying,
                i,j     bin fixed(15);

        i=0;
        do i=0 repeat (i+1)
                while(substr(string,(length(string)-i),1)=' ');
        end;
        string=substr(string,1,(length(string)-i));
        return (string);
        end strip;

        end maillab;
/* end of pl/i program */
```

The following listing shows a simple Microsoft Basic program to perform the same function of printing the console listing and mail labels upon the printer. Once again the logical program analysis is left to the reader. Note that in order to facilitate program comparisons, the logical program structure of the Basic program is kept nearly the same as the previous PL/I-80 program.

```
10 REM
20 REM
30 REM
40 REM
50 REM    TELL OPERATOR TO PUT LABELS INTO PRINTER
60 REM
70 PRINT:PRINT:PRINT
80 PRINT "Put mail labels into printer."
90 INPUT "Type 'GO' when ready....",NUL$
100 OPEN "I",#1,"NAMES.DAT"
110 REM
120 REM  SETUP FOR END OF FILE EXIT
130 REM
140 IF EOF(1)<0 THEN 470
150 REM
160 REM   READ FILE RECORD AND FORMAT DATA
170 REM
180 LINE INPUT #1,REC$
190 STRIP$=MID$(REC$,1,8)
200 GOSUB 530
210 LN$=STRIP$
220 STRIP$=MID$(REC$,9,8)
230 GOSUB 530
240 FNM$=STRIP$
250 MI$=MID$(REC$,17,1)
260 STRIP$=MID$(REC$,18,20)
270 GOSUB 530
280 AD$=STRIP$
290 STRIP$=MID$(REC$,38,11)
300 GOSUB 530
310 CTY$=STRIP$
320 ST$=MID$(REC$,49,2)
330 ZP$=MID$(REC$,51,5)
340 PH$=" ("+MID$(REC$,56,3)+") "+MID$(REC$,59,3)+"-"+MID$(REC$,
    62,4)
350 REM
360 REM PRINT NAME AND PHONE NUMBERS TO CONSOLE
370 REM
380 PRINT FNM$;" ";LN$;TAB( 30);PH$
390 REM
400 REM PRINT NAMES ON LABELS AT PRINTER
410 REM
420 LPRINT:LPRINT
430 LPRINT FNM$;" ";MI$;". ";LN$
440 LPRINT AD$
450 LPRINT CTY$;"  ";ST$;"  ";ZP$
460 GOTO 140
470 CLOSE 1
480 SYSTEM
490 END
500 REM
510 REM    SUBROUTINE TO STRIP EXCESS BLANKS OFF THE END OF A
    STRING
520 REM
530 IF MID$(STRIP$,LEN(STRIP$),1)<>" " THEN RETURN
540 STRIP$=MID$(STRIP$,1,LEN(STRIP$)-1)
550 GOTO 530
```

If either of the above programs is run, the output at the console appears as follows. A <cr> indicates operator data entry and carriage return key depression.

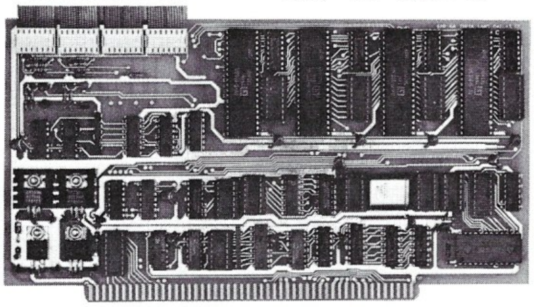
Console Output From Program

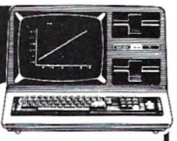
```
Put mail labels into printer.
Type 'GO' when ready....GO<cr>

MICHAEL KARAS                     (805) 527-7922
KELLY SMITH                       (805) 527-0518
JACOB JOHNSON                     (612) 342-4469
SHIELA ERICKSON                   (415) 443-5523
DALE WILLIAMS                     (217) 965-7451
MICHAEL HANSEN                    (805) 527-9355
HANSEN KARAS                      (305) 527-7922
GERALD NIXON                      (222) 555-1212
DEBBI PAULSON                     (398) 453-5551
A>
```

The following listing shows the printer output from either program. Only a portion of the listing is shown to give the idea without wasting too much paper. ■

Printer Output

```
MICHAEL J. KARAS
2468 HANSEN CT
SIMI VALLEY  CA  93065


KELLY S. SMITH
3055 WACO AVE
SIMI VALLEY  CA  93063


JACOB B. JOHNSON
2793 ANDREW COURT
MANKADO  MN  56705


SHIELA P. ERICKSON
454 B UNIVERSITY AVE
FARGO  ND  58102


DALE F. WILLIAMS
35912 CIRCLE MTN DR
DAVENPORT  IW  59034


MICHAEL H. HANSEN
2486 KARAS CT
SIMI VALLEY  CA  93065
```





Interfacing PL/I-80 with Assembly Language Programs

by Mark M. Zeiger

High level languages were invented to make the job of programming easier and faster. But as the saying goes—"If it's good, it's fattening," and thus with this ease of programming comes a certain loss in computer capability. The most obvious degradations are an increase in program size and usually a slower execution speed. The programmer also loses control over, or may not be able to make use of, certain machine functions. Some of the high level languages have tried to alleviate the latter problem; most Basics have "PEEK," "POKE," "INP," and "OUT" functions and statements. But most of the languages available under CP/M do not have the built-in capabilities for handling the primitive chores that the processor must sometimes perform. It is therefore necessary to write machine language routines to do these jobs and then to link them to the program written in the high level language. I would like to present some examples showing how I have done this with Digital Research's PL/I-80.

I have been using PL/I-80 for about a year, and like it better than most other high level languages. It has most of the structure of Pascal, and the I/O and print formatting is infinitely better. And believe it or not, the documentation that Digital Research supplies with PL/I-80 (the Link-80 manual in particular) is first-rate. I've heard complaints that you have to constantly re-read the manuals before understanding them, but I have never seen any computer software documentation where this is not true. Digital Research supplies all the utilities and information needed to link assembly language modules with PL/I-80 programs, and also supply a well-documented library of routines that will allow you to call upon the BDOS functions from a PL/I-80 program.

The routines which I have written allow PL/I-80 programs to perform some low level computer functions. I am able to access a clock-calendar board, input and output to and from I/O ports, address the cursor on a terminal, and perform certain console functions at a lower level than provided for in PL/I-80.

The key to interfacing external routines to any high level language is to understand how parameters are passed to a routine, and how results are returned to the calling program. When the PL/I-80 program calls any external routine, it enters the routine with the HL register pair containing the address of the first of a list of two byte addresses in contiguous locations. This second group of addresses are the memory locations where the parameters have been stored by the PL/I-80 program. For example, assume the routine "SAMPLE" has been declared as follows in a PL/I-80 program:

```
DECLARE SAMPLE ENTRY ( FIXED(7), FIXED(15), FIXED(7) );
```

where the "ENTRY" attribute informs the PL/I-80 compiler (and the Linker) that the routine SAMPLE is an external routine. SAMPLE might then be called with the following instruction:

```
CALL SAMPLE ( A, B, C );
```

where A and C were declared as fixed(7) and B as fixed(15). When sample is entered, the HL registers might contain 2000H. At 2000H the following data would be found in memory.

```
Reg. contents | address contents | address contents
              |                  |
HL    2000H   | 2000H    0030    | 3000H      A
              | 2002H    0130    | 3001H      BB    (2 bytes)
              | 2004H    0330    | 3003H      C
              |                  |
```

By examining the contents of address 2000H, you would find that the address where the first (single byte) variable "A" is stored is 3000H (remember that an 8080/Z-80 address is always low byte followed by high byte). The address of the next (double byte) variable "B" is stored at 2002H, so by examining the "word" at 2002H you would see that the variable "B" is stored at 3001H, and by examining the "word" at 2004H, you would find the third parameter to be stored in location 3003H. There are two things to note. First, by this scheme the addresses are expected to be stored in consecutive words, but the

Mark M. Zeiger, 198-01 B 67th St., Flushing, NY 11365.

NEW! TPM* for TRS-80 Model II
NEW! System/6 Package
Computer Design Labs

Z80* Disk Software

We have acquired the rights to all TDL software (& hardware). TDL software has long had the reputation of being the best in the industry. Computer Design Labs will continue to maintain, evolve and add to this superior line of quality software.
— Carl Galletti and Roger Amidon, owners.

Software with Manual/Manual Alone

All of the software below is available on any of the following media for operation with a Z80 CPU using the CP/M* or similar type disk operating system (such as our own TPM*).

for TRS-80* CP/M (Model I or II)
for 8" CP/M (soft sectored single density)
for 5¼" CP/M (soft sectored single density)
for 5¼" North Star CP/M (single density)
for 5¼" North Star CP/M (double density)

BASIC I

A powerful and fast Z80 Basic interpreter with EDIT, RENUMBER, TRACE, PRINT USING, assembly language subroutine CALL, LOADGO for "chaining", COPY to move text, EXCHANGE, KILL, LINE INPUT, error intercept, sequential file handling in both ASCII and binary formats, and much, much more. It runs in a little over 12 K. An excellent choice for games since the precision was limited to 7 digits in order to make it one of the fastest around. $49.95/$15.

BASIC II

Basic I but with 12 digit precision to make its power available to the business world with only a slight sacrifice in speed. Still runs faster than most other Basics (even those with much less precision). $99.95/$15.

BUSINESS BASIC

The most powerful Basic for business applications. It adds to Basic II with random or sequential disk files in either fixed or variable record lengths, simultaneous access to multiple disk files, PRIVACY command to prohibit user access to source code, global editing, added math functions, and disk file maintenance capability without leaving Basic (list, rename, or delete). $179.95/$25.

ZEDIT

A character oriented text editor with 26 commands and "macro" capability for stringing multiple commands together. Included are a complete array of character move, add, delete, and display function. $49.95./$15.

ZTEL

Z80 Text Editing Language - Not just a text editor. Actually a language which allows you to edit text and also write, save, and recall programs which manipulate text. Commands include conditional branching, subroutine calls, iteration, block move, expression evaluation, and much more. Contains 36 value registers and 10 text registers. Be creative! Manipulate text with commands you write using Ztel. $79.95/$25.

TOP

A Z80 Text Output Processor which will do text formatting for manuals, documents, and other word processing jobs. Works with any text editor. Does justification, page numbering and headings, spacing, centering, and much more! $79.95/$25.

MACRO I

A macro assembler which will generate relocateable or absolute code for the 8080 or Z80 using standard Intel mnemonics plus TDL/Z80 extensions. Functions include 14 conditionals, 16 listing controls, 54 pseudo-ops, 11 arithmetic/logical operations, local and global symbols, chaining files, linking capability with optional linker, and recursive/reiterative macros. This assembler is so powerful you'll think it is doing all the work for you. It actually makes assembly language programming much less of an effort and more creative. $79.95/$20.

MACRO II

Expands upon Macro I's linking capability (which is useful but somewhat limited) thereby being able to take full advantage of the optional Linker. Also a time and date function has been added and the listing capability improved. $99.95/$25.

LINKER

How many times have you written the same subroutine in each new program? Top notch professional programmers compile a library of these subroutines and use a Linker to tie them together at assembly time. Development time is thus drastically reduced and becomes comparable to writing in a high level language but with all the speed of assembly language. So, get the new CDL Linker and start writing programs in a fraction of the time it took before. Linker is compatible with Macro I & II as well as TDL/Xitan assemblers version 2.0 or later. $79.95/$20.

DEBUG I

Many programmers give up on writing in assembly language even though they know their programs would be faster and more powerful. To them assembly language seems difficult to understand and follow, as well as being a nightmare to debug. Well, not with proper tools like Debug I. With Debug I you can easily follow the flow of any Z80 or 8080 program. Trace the program one step at a time or 10 steps or whatever you like. At each step you will be able to see the instruction executed and what it did. If desired, modifications can then be made before continuing. It's all under your control. You can even skip displaying a subroutine call and up to seven breakpoints can be set during execution. Use of Debug I can pay for itself many times over by saving you valuable debugging time. $79.95/$20.

DEBUG II

This is an expanded debugger which has all of the features of Debug I plus many more. You can "trap" (i.e. trace a program until a set of register, flag, and/or memory conditions occur). Also, instructions may be entered and executed immediately. This makes it easy to learn new instructions by examining registers/memory before and after. And a RADIX function allows changing between ASCII, binary, decimal, hex, octal, signed decimal, or split octal. All these features and more add up to give you a very powerful development tool. Both Debug I and II must run on a Z80 but will debug both Z80 and 8080 code. $99.95/$20.

ZAPPLE

A Z80 executive and debug monitor. Capable of search, ASCII put and display, read and write to I/O ports, hex math, breakpoint, execute, move, fill, display, read and write in Intel or binary format tape, and more! on disk $34.95/$15.

APPLE

8080 version of Zapple $34.95/$15.

NEW! TPM now available for TRS-80 Model II!

TPM*

A NEW Z80 disk operation system! This is not CP/M*. It's better! You can still run any program which runs with CP/M* but unlike CP/M* this operating system was written specifically for the Z80* and takes full advantage of its extra powerful instruction set. In other words its not warmed over 8080 code! Available for TRS-80* (Model I or II). Tarbell, Xitan DDDC, SD Sales "VERSA-FLOPPY", North Star (SD&DD), and Digital (Micro) Systems. $79.95/$25.

SYSTEM MONITOR BOARD (SMB II)

A complete I/O board for S-100 systems. 2 serial ports, 2 parallel ports, 1200/2400 baud cassette tape interface, sockets for 2K of RAM, 3-2708/2716 EPROM's or ROM, jump on reset circuitry. Bare board $49.95/$20.

ROM FOR SMB II

2KX8 masked ROM of Zapple monitor. Includes source listing $34.95/$15.

PAYROLL (source code only)

The Osborne package. Requires C Basic 2.
5" disks $124.95 (manual not included)
8" disks $ 99.95 (manual not included)
Manual $20.00

ACCOUNTS PAYABLE/RECEIVABLE (source code only)

By Osborne, Requires C Basic 2
5" disks $124.95 (manual not included)
8" $99.95 (manual not included)
Manual $20.00

GENERAL LEDGER (source code only)

By Osborne. Requires C Basic 2
5" disks $99.95 (manual not included)
8" disks $99.95 (manual not included)
Manual $20.00

C BASIC 2

Required for Osborne software. $99.95/$20.

SYSTEM/6

TPM with utilities, Basic I interpreter, Basic E compiler, Macro I assembler, Debug I debugger, and ZEDIT text editor.
Above purchased separately costs $339.75
Special introductory offer: Only $179.75 with coupon!!

$160.

This Coupon Is Worth One Hundred And Sixty Dollars Toward The Full Price Of The SYSTEM/6 Package System/6 with this coupon is only $179.95. This is a limited time offer.

$160.00

ORDERING INFORMATION

Visa, Master Charge and C.O.D. O.K. To order call or write with the following information.

1. Name of Product (e.g. Macro I)
2. Media (e.g. 8" CP/M)
3. Price and method of payment (e.g. C.O.D.) include credit card info. if applicable.
4. Name, Address and Phone number.
5. For TPM orders only: Indicate if for TRS 80, Tarbell, Xitan DDDC, SD Sales (5¼" or 8"). ICOM (5¼" or 8"), North Star (single or double density) or Digital (Micro) Systems.
6. N.J. residents add 5% sales tax.

Manual cost applicable against price of subsequent software purchase in any item except for the Osborne software.

For information and tech queries call
609-599-2146

For phone orders ONLY call toll free
1-800-327-9191
Ext. 676
(Except Florida)

OEMS

Many CDL products are available for licensing to OEMs. Write to Carl Galletti with your requirements.

* Z80 is a trademark of Zilog
* TRS-80 is a trademark for Radio Shack
* TPM is a trademark of Computer Design Labs. It is not CP/M*
* CP/M is a trademark of Digital Research

Prices and specifications subject to change without notice.

DEALER INQUIRIES INVITED.

COMPUTER
DESIGN
LABS

342 Columbus Avenue
Trenton, N.J. 08629

variables could be stored in any locations in memory. Usually the variables would be stored in consecutive locations (especially if they are of similar types), but this is not guaranteed by the parameter-passing conventions. Second, the parameter-passing conventions in no way indicate the number of bytes associated with each value. This must be known and accounted for when the programmer formulates his routine.

Upon entering a routine you will usually want to get the parameters and do what must be done with them. Below is a routine which will put the address of the first parameter in the DE register pair when it is initially called (i.e., the DE pair would contain 3000H in this example) and then likewise the address of the next parameter upon subsequent calls to the routine. This routine should be part of every assembly language subroutine that is called by a PL/I-80 program if the routine is to pass two or more parameters.

```
GET$PAR$ADDR:  MOV  E,M  ;move low byte of address to E-reg
               INX  H    ;point to second byte of address
               MOV  D,M  ;put high byte of address in D-reg
               INX  H    ;point to address of next parameter
               SHLD ADDRESS$SAVE   ;save address for next call
               RET       ;with address of parameter in DE pair
ADDRESS$SAVE:  DS   2
```

An entire sequence would be as follows:

```
SAMPLE:          CALL GET$PAR$ADDR   ;address of 1st par in DE
                 LDAX D              ;1st par in A-reg
                   .
                   .                 ;do what you want with it
                   .

                 LHLD ADDRESS$SAVE   ;get pointer to addr of next par
                 CALL GET$PAR$ADDR   ;pointer to parameter in DE
                   .
                   .                 ;do what you want with this
                   .                 ;two byte number

                 LHLD ADDRESS$SAVE   ;get pointer to addr of 3rd par
                 CALL GET$PAR$ADDR   ;pointer to parameter in DE
                   .
                   .                 ;do what you wish
                   .

                 RET                 ;return to PL/I program
```

Notice that this routine does not return any values to the PL/I-80 program. Many times however, a machine language routine must do so. Most of the conventions for returning values to the calling program are quite easy to implement. The PL/I-80 program expects all one-byte values (except single characters) to be returned in the A-register. This includes variables declared as fixed(1) to fixed(7) and bit(1) to bit(8). All two-byte values—such as pointers, label and entry variables, fixed(8) to fixed(15), and bit(9) to bit(16)—are returned in the HL register pair with the A-register being set equal to the L-register (the latter does not seem to be mandatory even though the Link-80 manual states it should be done). Character strings, fixed decimal numbers, and floating point numbers will be discussed later, since returning these types to a PL/I-80 program is more complicated.

Let's say we want a PL/I-80 program to call a routine that adds two numbers and returns the answer for PL/I-80 to output. It's a ridiculous example since PL/I-80 is quite capable of doing its own addition, but it's an easy example to follow.

The PL/I-80 program:

```
ADD:       PROCEDURE OPTIONS (MAIN);

DCL        ( A, B )  FIXED(7),              /* the addends */
           C         FIXED(15);             /* the result */

DCL        SUM ENTRY ( FIXED(7), FIXED(7) ) RETURNS (FIXED(15) );

A = 12;
B = 24;
C = SUM(A, B);
PUT LIST (C);

END ADD;
```

The assembly language program (the function SUM) would be:

```
PUBLIC    SUM                ;this serves to notify the linker that the
                             ;label "SUM" will serve as the entry
                             ;point of an externally called routine.

SUM:      CALL GET$PAR$ADDR  ;defined above
          LDAX D             ;put 1st one byte parameter in A-reg
          MOV  B,A           ;and save it in B-reg
          LHLD ADDRESS$SAVE  ;get address of pointer to next paramete
          CALL GET$PAR$ADDR  ;and get address of next parameter in DE
          LDAX D             ;put 2nd number in A (1st is in B)
          MOV  E,A           ;prepare to add with DAD instruction
          MVI  D,0           ;since result might be two bytes
          MOV  L,B           ;1st number in HL
          MVI  H,0
          DAD  D             ;sum in HL
          MOV A,L            ;duplicate L in A
          RET                ;return to PL/I with two byte value in H

GET$PAR$ADDR:
          MOV  E,M           ;described above
          INX  H
          MOV  D,M
          INX  H
          SHLD ADDRESS$SAVE
          RET

ADDRESS$SAVE:  DS 2

          END
```

To get this program to work, the source of the PL/I-80 program must be compiled into a REL file using PLI.COM. The source of the assembly language file must be assembled into a REL file by RMAC.COM (supplied by Digital Research with the PL/I-80 package). The two REL files are then linked to produce an executable object file using LINK.COM.

Assume that the source file is in the CP/M directory as a file called PART1.PLI. This will be compiled into PART1.REL by the PL/I compiler. If the assembly language program is in the file PART2.ASM, it will be assembled into PART2.REL by RMAC. The linker may then be used to generate the object file TEST.COM by the following command:

```
LINK TEST=PART1,PART2[S]
```

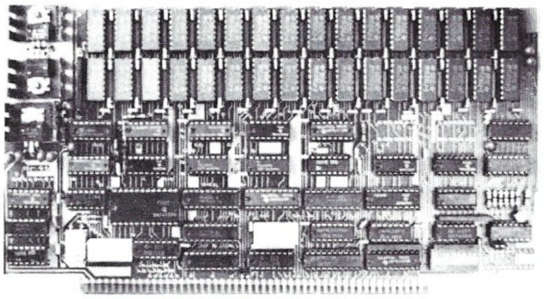
The [S] tells the linker to use only those modules requested by the PL/I-80 program. It is a useless command here since PART2.REL has only one module, but many times it is possible for a library file to contain several subroutines. Unless the [S] switch is used, every module in the library will be linked to the PL/I-80 program whether the PL/I-80 program requires it or not.

The example I used was meaningless, but I'm sure the reader will realize the power of the above procedure. It is only necessary to write a routine and compile it once. A number of different routines may be saved in one REL file (they may be combined by using Digital Research's LIB.COM) and then linked to a program each time they are needed. I have made up about ten routines (some in assembly language and some in PL/I-80) that I consistently use in each program I write. Instead of including them in the PL/I-80 source file, I just declare them at the beginning of the program and use the linker to get them. I have even made up a header file (similar to what is done in "C") which has all the declarations I might need. This is included in each source file by using the PL/I-80 command "%include 'A:HEADER.PLI';". The declared routines will not be linked unless they are actually used in the program, so no memory or disk space will be wasted by declaring unused routines.

The full PL/I-80 language has provisions for getting the date and time from whatever peripheral device is keeping track of such things, but there are no commands to do this in Subset-G of PL/I-80. (Subset-G is the standard

256K RAM IN 4K BLOCKS

1/4 Megabyte $1495



OVERVIEW

The BSR 64/256 is an 8 bit bank selectable dynamic random access memory card designed to operate in a Z-80 based S-100 computer system with a CPU clock frequency of up to 4 MHz(A model) or 6 MHz(B model).

Individual 64K banks are selected via the IEEE 696 8 bit address us extension. If the host system is not capable of driving the extended address bus, one of the BSR 64/256 cards in the system may be configured to drive it through an onboard latched output port.

System area is allocated in 4K blocks by writing a system mask out to two latched output ports. Another port allows any one of up to eight cards to be assigned as the current system master. Logically, up to 64 cards may be addressed in a single computer system.

Assembled & Tested Boards

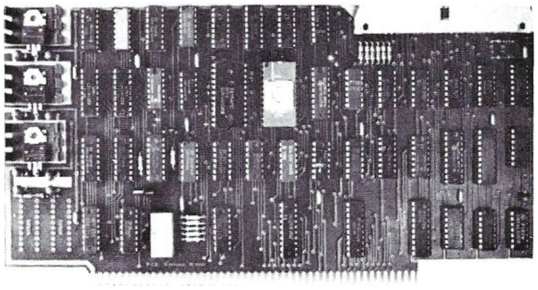
(4164)	64k	$770
	128k	$1015
	192k	$1260
	256k	$1495

Memory prices fluctuate, call for price

FEATURES

MAX STORAGE	256k bytes (32 ea. 64k X 1 chips) or 64k bytes (32 ea. 16k X 1 chips)
BLOCK SELECTION	Any combination of 4k blocks in any 64k bank, software selectable
SYSTEM AREA	Any combination of 4k blocks in low order bank of current system master card, software selectable
SYSTEM MASTERS	Any one of up to 8 cards software selectable, one card jumper selectable for system power-up or reset
CARDS/SYSTEM	Logically, up to 64 64k or 256k cards
BANK SELECTION	Uses or implements IEEE 696 (S-100) extended address bus
CPU SPEED	4 MHz or 6 MHz with no wait states
REFRESH MODE	Invisible
OUTPUT PORTS	4 Consecutive ports for entire system, selectable on any 4 port boundary

14″ WINCHESTER FIXED DISK CONTROLLER & SUBSYSTEM



FDC 4000

14″ Winchester Fixed Disk Controller

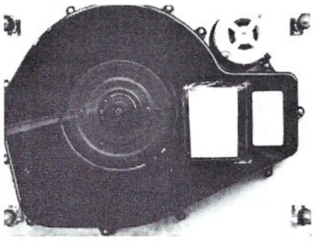
Controls Shugart 4000 & 4100 series drives
Up to 174 megabytes per controller
I/O mapped buffer - no wait states
Includes MP/M* XIOS & CP/M* CBIOS

Assembled & tested $895

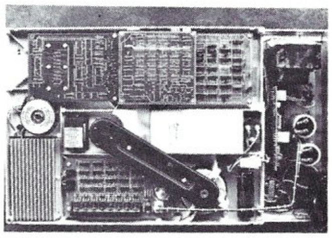
29 Megabyte Disk Subsystem $3840

The fixed disk subsystem includes a 29 million byte Shugart fixed disk (SA 4008), and an assembled and tested S-100 disk controller (FDC 4000). Also included is an unfinished rack mountable aluminum case, punched and ready for assembly. To complete this kit we have included power supply, fan, wiring harness, signal cable, and MP/M* XIOS and CP/M* CBIOS.

(SA 4008)



(Assembled Subsystem)



PCE SYSTEMS

"Contact Us For More Information"

5232 Manzanita Ave. Carmichael Ca 95608 (916) 338-5454

Price subject to change

*Trademark of Digital Research

which Digital Research used in writing the compiler.) Since I have a clock-calendar, I decided I wanted this feature, and wrote the routine to return a date-time ASCII string to a PL/I-80 program. This is how I became interested in writing assembly language routines which could be called by a high level language. Since I wrote the first TIME routine, I have learned a few neat tricks to make the interfacing much easier. However, I am going to include my first effort because it is a good way to show how to return character strings to PL/I-80 from a machine language routine.

We saw before that it was fairly easy to return a one- and two-byte value to PL/I-80 by using the accumulator or HL register pair. Returning character strings (even one character), decimal values, or floating point values is more involved. All three types must be returned on the stack with the most significant digit or first character at the top of the stack. The routine must also put the length of the character string in the A-register, while in the case of a fixed decimal number the PL/I-80 program will expect sixteen digits (eight bytes) to be returned regardless of the actual precision of the number. In the case of a floating point number, PL/I-80 expects four bytes containing the mantissa and the exponent to be on the stack. The Link-80 manual does a good job of describing the format of decimal and floating point numbers. The first thing to do upon entering a routine which return values on the stack is to POP the return address off the stack and save it in memory, since you won't be able to use the stack for the RET instruction. You may then PUSH the string or number onto the stack, load the HL registers with the return address, which was previously saved, and use a PCHL instruction to return to PL/I-80.

The following clock routine expects a one-byte[fixed (7)] parameter from PL/I-80 and returns a certain string depending upon the value of the number passes. Assuming that it is October 14, 1981 at 4:51:21 PM, then:

```
if 1 is passed, the routine returns "16:51:21"
if 2 is passed, the routine returns "14-Oct-81"
if 3 is passed, the routine returns "14-Oct-81bbb16:51:21"
                                                              b = space
if any other value is passed, the program will print an
error message and return to CP/M (this should never happen
except in the case of sloppy programming, but it's a good
check).
```

In PL/I-80 the TIME routine would be declared as:

```
DECLARE TIME ENTRY ( FIXED(7) ) RETURNS ( CHAR(20) VARYING );
```

and it may be implemented by instructions such as:

```
PUT SKIP LIST ( TIME(2) );    or maybe
TIME-OF-DAY = TIME(1);        where TIME-OF-DAY is a char(8) variable.
```

The following is a well-commented listing of the 8080 TIME routine:

```
      PUBLIC    TIME      ;entry point to routine

TIME: POP  D              ;get return address off stack and..
      XCHG                ;..save it for later return. Note that..
      SHLD STKSV          ;..HL regs must be preserved to find parameters
      XCHG                ;get back address of parameter pointer table
      CALL GET$PAR$ADDR   ;defined above
      LDAX D              ;get parameter in A-reg and..
      STA  FSAVE          ;..save it for later.


*******************************************************************
*                                                                 *
*                   ROUTINE FOR GETTING TIME                      *
*                                                                 *
*   This section depends upon your clock-calendar board. After the *
*   date and time are calculated, an ASCII string should be con-  *
*   structed that looks as follows:                               *
*                                                                 *
*                 14-Oct-81bbb16:51:21                            *
*                                                                 *
*   with the HL register pair containing the address of the string *
*   (the leftmost byte). Naturally the format may be different, but *
*   then the rest of the program will have to be changed slightly  *
*   to suit the format.                                           *
*                                                                 *
*   In my particular case, the clock access routines are in my    *
*   BIOS of CP/M 2.2. Thus by using function 38 (which is unused  *
*   by CP/M) I am able to format the date and time in various     *
*   ways (depending upon the value in the E-reg). This procedure  *
*   was developed by Harvey Fishman.                              *
*                                                                 *
*******************************************************************


;routine for returning string

      LDA   FSAVE         ;get back initial parameter
      CPI   1
      JNZ   DATEP         ;if not 1, then see if 2 or 3

                          ;It is a 1, therefore an 8 byte string
                          ;consisting of the time will be returned.
                          ;Thus 4 pushes are needed (2 bytes/push)
                          ;and the first character pushed will be
                          ;the last character in the time portion
                          ;of the string.
      MVI   A,4           ;number of pushes
      LXI   D,19          ;first char to be pushed is 19 bytes from
                          ;beginning of string
      JMP   DOPUSH        ;go and do pushes

DATEP:
      CPI   2
      JNZ   DTP           ;if not a 2, then check if 3
      MVI   A,5           ;it's a 2. Get date string which is 9 chars
                          ;long. Therefore 5 pushes. Last char of date
                          ;string is located 8 chars from beginning of
      LXI   D,8           ;string
      JMP   DOPUSH        ;do the pushes

DTP:  CPI   3
      JNZ   ERROR         ;this should never happen once the PL/I
                          ;program is debugged
      MVI   A,10          ;if 3 get whole string. 20 chars -> 10 pushes
      LXI   D,19          ;last char is 19 from beginning

DOPUSH:
      MOV   B,A           ;count pushes in B-reg
      ADD   A             ;double A-reg to get # chars in string
      DAD   D             ;address first char to be pushed by HL
```

```
LOOP:
      MOV  D,M              ;put last two characters in DE reg
      DCX  H                ;in order to push them
      MOV  E,M
      DCX  H                ;point to next pair of chars
      PUSH D                ;push two characters in DE on stack
      DCR  B                ;count pushes
      JNZ  LOOP             ;do another push

                            ;string is now on stack and A-reg contains
                            ;number of characters to be returned.

      LHLD STKSV            ;get return address saved at beginning

      PCHL                  ;back to PL/I

ERROR:
      MVI  C,9              ;print message and reboot CP/M. This error
      LXI  D,ERRMSG         ;should never happen in debugged program.
      CALL BDOS
      RST  0                ;jump to zero for warm boot

GET$PAR$ADDR:               ;defined above
       .
       .
       .
      RET


STKSV:     DS    2
FSAVE:     DS    1
BDOS       EQU   5
ERRMSG:    DB    'Illegal TIME parameter$'

      END
```

A short time after I created the TIME routine, I decided I wanted it to be more versatile. The actual output of the board is a series of thirteen bytes which represent the following:

1) tens digit of last two digits of year
2) units digit of last two digits of year
3) tens digit of month
4) units digit of month
5) tens digit of day
6) units digit of day
7) day of week   0 = Sunday
8) tens digit of hour
9) units digit of hour
10) tens digit of minute
11) units digit of minute
12) tens digit of second
13) units digit of second

Therefore the output for the date used in the last problem would be the binary values 8 1 1 0 1 4 3 1 6 5 1 2 1 in contiguous memory locations.

These bytes could be put into an array of dimension thirteen. The Link-80 manual gives no indication as to how to pass an entire array of structure between routines, but I think that an easy way of doing this is by declaring the array as a BASED variable. Then all that would have to be done is return the address of the first byte of the array in the HL register pair. This same technique would work when returning the date-time string in the last procedure, and the actual process of returning the data is much simpler. To perform the procedure with based variables, first declare the following in PL/I-80:

```
DECLARE ARRAYPTR POINTER,
        DT (13) FIXED(7) BASED (ARRAYPTR),
        STRINGPTR POINTER,
        TIME-STRING CHAR(20) BASED (STRINGPTR);        /* MUST not be
                                                         char VARYING */
DECLARE PTIME ENTRY (FIXED(7) ) RETURNS (POINTER);
```

If the parameter passed by PL/I-80 is one, it tells the clock routine to return the ASCII character string used. If the parameter is two, only the binary digits are formatted. In each case a pointer to either the first byte of the string or to DT(1) is returned. The routine could be called by the following types of instructions:

```
DCL   (MONTH, DAY, YEAR) FIXED(7);
ARRAYPTR = PTIME(2);          /* calculates time and returns pointer */
MONTH = 10*DT(3) + DT(4);
DAY = 10*DT(5) + DT(6);
YEAR = 10*DT(1) + DT(2);
PUT EDIT ( MONTH, '/', DAY, '/', YEAR ) (F(2), A, F(2), A, F(2) );
```

The output would be 10/14/81.

If we wanted just the time string returned we could do the following:

```
STRINGPTR = PTIME(1);         /* calculates time and returns pointer */
PUT EDIT (SUBSTR(TIME-STRING), 13,20);
```

The output would be 16:51:21.

While the PL/I-80 program looks much more complicated when using pointers, the corresponding assembly language routine is simpler to write. This is true because only the pointer need be returned in the HL registers, and no bytes have to be pushed on the stack. This technique could also be used to return fixed decimal and floating point numbers. The assembly language routine would be:

```
    PUBLIC    PTIME

PTIME:    CALL GET$PAR$ADDR
          LDAX D


*****************************************************************
*                                                               *
*    Machine dependent routine to get date and time, and:      *
*         1.   put in contiguous bytes if A-reg contains 2     *
*         2.   format as an ASCII string if A-reg contains 1   *
*                                                               *
*    HL register pair must point towards either first byte of  *
*    array or first character in string. It would be a routine *
*    similar to the one used in the last TIME routine. Harvey's *
*    function 38 does it automatically.                         *
*                                                               *
*****************************************************************


          RET

GET$PAR$ADDR:                   ;defined above
           .
           .
           .
          RET
ADDRESS$SAVE: DS 2

          END
```

Interfacing PL/I-80, continued...

Based variable and pointers may be used in other ways to take advantage of some of the conveniences of CP/M. In CP/M, programs may be called using the command line conventions. Anything typed after the program name is put into a buffer starting at location 81H (the number of characters typed is put into 80H). To access the string typed after calling a program created by the PL/I-80 compiler and linker, use a based character string variable whose pointer has the value of 80H. The program would be:

```
DECLARE PTR POINTER,
        STRING CHAR(127) VARYING BASED (PTR);   /* must be VARYING */

PTR = UNSPEC('0080'B4) /* puts the address 80H into the pointer */
                       /* variable. The UNSPEC function is not  */
                       /* a standard PL/I routine. The 80H must */
PUT EDIT (STRING) (A); /* be formatted with leading zeros and   */
                       /* the "B4" indicates the number is in   */
                       /* HEX.                                  */
```

The following are some other routines I regularly use and have put into the main PL/I-80 library (PLILIB.IRL).

The following routine addresses a cursor (ADM-3A terminal):

```
     PUBLIC    CURSOR

CURSOR:   CALL GET$PAR$ADDR
          LDAX D              ;first parameter is row number
          ADI  20H            ;ADM-3A has row offset of 32
          MOV  B,A            ;save row number
          CALL GET$PAR$ADDR   ;no need to reload with ADDRESS$SAVE
                              ;since HL regs were not touched.
          LDAX D              ;get column
          ADI  20H            ;offset of 32 also
          MOV  C,A            ;save column in C-reg
          MVI  A,27           ;ASCII escape
          CALL CONOUT         ;send to terminal
          MVI  A,'='          ;2nd byte of cursor positioning str
          CALL CONOUT         ;send it
          MOV  A,B            ;get row in A-reg
          CALL CONOUT         ;send it
          MOV  A,C            ;get column
          CALL CONOUT
          RET                 ;back to PL/I

CONOUT:   LHLD 1              ;bios entry point
          LXI  D,9            ;offset for conout jump
          DAD  D              ;address of conout jmp in HL
          PCHL                ;jump to conout jump. Return done
                              ;from BIOS to instruction just after
                              ;CALL CONOUT instruction

GET$PAR$ADDR:                 ;described above
            .
            .
            .
          RET

          END
```

In PL/I-80 the routine is declared as:

```
DCL  CURSOR ENTRY ( FIXED(7), FIXED(7) );
```

and called by:

```
CALL CURSOR(14,55);
```

Direct console input and status checks may be used to avoid some of the pitfalls of using PL/I-80 input routines. The "GET LIST" or "GET EDIT" routines always echo what is type, and will echo control characters in the same manner as the CP/M command line. Also, a control-C at the beginning of a line will cause a warm boot—a disastrous result if you have been entering a great deal of data to a file and the file has not been closed. The following routines avoid these problems.

```
     PUBLIC     CONIN, CONSTAT ;2 possible entry points in
                               ;this module

CONIN:                         ;returns char typed without echo.
                               ;will not echo control chars
     CALL GOTTOBIOS            ;jump to BIOS conin routine
                               ;returns with input in A-reg
     POP  H                    ;get returns address to PL/I
     PUSH PSW                  ;push conin char onto stack
     INX  SP                   ;increase stack point to account
                               ;for flags being pushed on stack
                               ;Routine taken from LINK-80 manual
     MVI  A,1                  ;only one char being returned
     PCHL                      ;return address to PC - ret to PL/I
GOTOBIOS:
    LHLD 1                    ;address of BIOS jump table
    LXI  D,6                  ;offset for conin jump
    DAD  D                    ;HL points to conin jump
    PCHL                      ;go there- return done from BIOS

CONSTAT:                      ;second module
                              ;returns 0 if key not pressed
                              ;returns -128 if key pressed
    LHLD 1                    ;addr of BIOS jump table
    LXI  D,3                  ;constat offset
    DAD  D                    ;HL points to constat jump
    PCHL                      ;go to constat jump. RETurn of bios
                              ;returns to PL/I with status value
                              ;in A-reg. Constat should return a
                              ;fixed(7) number.

    END
```

To use the above routines from PL/I:

```
    DECLARE   CONIN ENTRY RETURNS (CHAR(1) ),
              CONSTAT ENTRY RETURNS (FIXED(7) ),
              LETTER CHAR(1);

    LETTER = CONIN();          /* function call must have parentheses
                                  even if no parameters are passed. */
    PUT EDIT (LETTER) (A);     /* to echo conin character           */

    IF CONSTAT() ^= 0 THEN CALL PAUSE;/* will cause pause if any
                                         key is pressed            */

PAUSE:    PROCEDURE;
DCL       DUMMY CHAR(1);
          DUMMY = CONIN();    /* will wait until key pressed
                                 no character is echoed          */
END PAUSE;
```

The conin routine may only be used to get a single character, but I have written a PL/I-80 routine that will allow inputting a string of up to 128 characters without having the problem of control characters being echoed. The routine uses the CONIN procedure. Like an assembly language program, an independent PL/I-80 procedure may be compiled by PLI.COM into a REL file and then inserted into a separate library or the main library supplied by Digital Research. In fact, it is easier to write the PL/I-80 routine than to write an equivalent assembly language routine since PL/I-80 handles all the parameter passing. I have included a listing of the procedure (called INPSTR) at the end of this article.

The next two routines allow input and output to ports. They are similar to Basic's INP and OUT commands. In order to get values up to 255, a fixed(8) integer must be used. Any integer above fixed(7) uses two bytes; in a fixed(8) value the high byte contains only the sign bit. The sign bit will be intentionally ignored when passed to the routine, and made zero before returning to PL/I-80. Therefore it will appear that we are always dealing with a non-negative integer from 0 to 255.

```
     PUBLIC     INP, OUTP      ;could not use "OUT" since the assembler
                               ;interpreted it as an instruction

OUTP:
     CALL GET$PAR$ADDR         ;get address of first byte of a two byte
                               ;integer. This is the low byte which
                               ;contains the number. The high byte
                               ;contains only the sign bit, so we will
                               ;ignore it.
     LDAX D                    ;the first parameter is the port number
                               ;the second is the value to be outputted
     STA  OUTPNUM              ;patch the OUT instruction with port #
     LHLD ADDRESS$SAVE         ;get byte to output
     CALL GET$PAR$ADDR
     LDAX D                    ;put it in A-reg
     OUT  DUMMY                ;patched by STA instruction above
OUTPNUM EQU $-1                ;OUTPNUM is 2nd byte of OUT instruction
     RET                       ;return to PL/I

INP: CALL GET$PAR$ADDR         ;get 1st (and only) parameter - the port #
     LDAX D                    ;port number in A-reg
     STA  INPNUM               ;patch IN instruction
     IN   DUMMY                ;2nd byte patched by STA instruction
INPNUM EQU $-1
     MOV  L,A                  ;two byte numbers returned in HL regs
     MVI  H,0                  ;sign bit in H - make number positive
     RET                       ;so that it will be between 0 and 255

     DUMMY     EQU 0

GET$PAR$ADDR:                  ;defined before
        .
        .
        .
     RET

     END
```

To use the INP function in PL/I:

```
DCL  INP ENTRY (FIXED(8)) RETURNS (FIXED(8)),
     BYTE FIXED(8);                /* really uses 9 bits */

BYTE = INP(6)              /* will input from port 6 and put returned
                              value in the variable BYTE              */
```

To use the OUTP routine:

```
DCL  OUTP ENTRY (FIXED(8), FIXED(8)),
     (PORTNUM, VALUE) FIXED(8);

PORTNUM = 255; VALUE = 6;
CALL OUTP((PORTNUM, VALUE);     /* will output 6 to port 0FFH */
```

And finally, one simple routine which allows you to end a PL/I-80 routine gracefully. I have always disliked the message "End of execution" when a PL/I-80 program ends, so I wrote the routine STOPPGM. But you must be careful when using this routine because it will not close files automatically as will the STOP command of PL/I-80.

```
          PUBLIC     STOPPGM

STOPPGM:  RST  0

          END
```

and in PL/I:

```
     DCL  STOPPGM ENTRY;
     CALL STOPPGM;
```

I hope I have given you some idea of how you can make PL/I-80 an even more powerful language by calling upon machine language routines. While I have tried to go into as much detail as possible, you should read the Link-80 manual carefully. The examples given there, supplemented with the ones in this article, should give you a good idea as to how to use the link feature. Be sure to read the documentation on LIB.COM to see how to put all the external routines you write into one library. And don't be afraid to experiment—it's the only way to learn. ■

Listing of INPSTR

```
inpstr:         procedure (delimiter) returns (char(128) varying);

/* If 0 is passed to procedure then only C/R is accepted as the
   string terminator. If 1 is passed, then "escape" is accepted
   as terminator as well. Typing an "escape" will cause the routine
   to return with a string of length 1 containing only an "escape"
   regardless of how many characters were previously entered.

   The procedure is declared in the PL/I program as:

     DECLARE INPSTR ENTRY (FIXED(7)) RETURNS (CHAR(128) VARYING);

   andifput into a library, must be placedbeforethe CONIN
   routine which it uses.

   Used in PL/I as:

   STRING = INPSTR(0);   where STRING is a char(128) varying variable.

*/

%replace true by '1'b, false by '0'b;

declare         delimiter fixed(7),
                input-string char(128) varying,
                input-char char(1),
                conin entry returns (char(1));

        input-string = '';
        do while (true);
                input-char = conin();
                if delimiter = 1 then
                   if input-char = ascii(27) then return (input-char);
                if input-char = ascii(13) then do;
                        put skip;
                        return (input-string);
                end; /* if input-char = */

          /* routines for deletion  */

                if (input-char = ascii(8) | input-char = ascii(127) ) &
                        length(input-string) > 0 then do;
                  input-string = substr(input-string, 1,
                        length(input-string) - 1);
                  put edit (ascii(8), ' ', ascii(8)) (3a);
                end; /* if input-char = */

                if input-char >= ' ' & input-char <= ascii(126) then do;
                        input-string = input-string || input-char;
                        put edit (input-char) (a);
                end; /* if input-char = */

        end; /* do while (true) */
  end inpstr;
```

The PL/I-80 Language

by Andrew Bender

A close look at Digital Research's new PL/I compiler.

It is difficult to write an extensive review of new software without providing the reader with some background information. For the casual user of a language this information forms a starting point from which more can be learned about the features of the language. For others, some of the design philosophies may be more easily understood. In October of 1963 a few managers of IBM and members of its Fortran compiler team met with key members of the IBM scientific computer users group SHARE to form the Advanced Language Development Committee of the SHARE FORTRAN Project.

The IBM New Product Line, later to become the S/360, was to be released with a New Programming Language. Hence, the first acronym, NPL. Since NPL conflicted with the established abbreviation for the National Physical Laboratory in England, it was changed to PL/I. PL/I was to incorporate the best features of Fortran, and become an extension of Fortran. It did not take the committee long to realize that many of the restrictions in the Fortran language did not allow for a reasonable extension. As a result of this realization and over strong objections, the new PL/I language took shape on its own rather than being another Fortran.

In March of 1964 the new PL/I language was presented to the SHARE group, and it was both praised for its scope and criticized for its complexity. GUIDE, which was to business users what SHARE was to scientific users, appointed a member to the committee because the language was to have commercial appeal as well. The document describing the language was presented at the meeting, and soon after the language was compared with Cobol, Jovial and Fortran. The conclusion was "...NPL is a very strong and powerful language...." In fact, considering the fact that the language was designed over fifteen years ago, it contains many of those elements deemed desirable today:

- Production of well-structured programs
- Ability to prove correctness
- Interactive language facilities
- Extensible
- Solid theoretical basis.

Looking critically at the languages recently proposed, will we be able to make the same comments in fifteen years about these new languages? With this in mind, let us look critically at PL/I-80.

I tested the PL/I-80 package which is designed to run on 8080, 8085 or Z-80 based 8-bit systems. The following software is included in the package:

- PL/I-80 Compiler
- RMAC Relocating Macro-Assembler
- LINK-80 Linkage Editor
- LIB Run-Time Library

PL/I-80, as released in version 1.3 by Digital Research, consists of five manuals and some addenda and notes, as well as two 8" floppy disks in CP/M format. One manual is the Digital Research PL/I language specification. This is written in specification style—very tough reading. You really need a background in PL/I programming to use this manual. A more readable manual is the pragmatic PL/I "Applications Guide." This manual teaches by example, and is limited by however many examples of such a powerful language can be given. This is not to be taken as any reflection on Digital Research. They have done more than most vendors by taking a pragmatic approach to a subject usualy covered only by a specification manual.

The third manual gives a capsule summary of PL/I and also lists the error messages and their meanings. This little booklet was evidently not distributed with earlier

Andrew Bender, Neurological Services, Inc., 336 Center Ave., Westwood, NJ 07675.

OTHERS MAY SEE THE ERRORS OF YOUR WAYZ.

ONLY MICROSPELL CORRECTS THOSE WAYS.

Lifeboat Associates, the world's foremost source for microcomputer software, proudly presents MicroSpell,™ the first program that not only isolates spelling errors in your text, but actually corrects them.

MicroSpell works with your word processor. And if you have the best word processing system, why settle for anything less than the *best* spelling corrector?

Goes Beyond The Competition

Other spelling programs function primarily as spelling checkers, merely pointing out words with suspect spelling. It's left up to you to determine the correct spelling and then type it in.

MicroSpell, the only spelling program that knows how to "spell," corrects the error automatically. Here's how it works: MicroSpell will read your text, carefully looking for words that might be spelled incorrectly. When it comes across a word that it's not absolutely sure of, it stops and shows you that word, along with its context. Then it searches through its own built-in dictionary and presents a list of guesses which it "thinks" might be correct. All you have to do is press a key and the misspelled word is corrected. There is no need to bother with your own dictionary, or even to type in the change. MicroSpell will do it all for you!

The Most Complete Built-In Dictionary

MicroSpell uses word stems and suffix stripping routines, so its dictionary of 25,000 word-parts can deliver over 150,000 words to you. And if that's not enough, MicroSpell will let you add thousands of additional words, so you can create and store specialized dictionaries of technical terms, unusual expressions, even acronyms. And you can let MicroSpell know just when any of these special dictionaries are wanted.

MicroSpell is highly interactive and designed to complement word processing systems that create ASCII text using a CP/M® compatible operating system. It requires minimum disk storage capacity of 70K per drive.

So why settle for a program that merely finds your *mispellings* when you can get the one that corrects your *misspellings*? MicroSpell. It corrects the errors of your ways.

MicroSpell is brought to you exclusively and supported completely by Lifeboat Associates. Call or send us the coupon below.

LIFEBOAT WORLDWIDE offers you the world's largest library of software. Contact your nearest dealer or Lifeboat:

Lifeboat Associates
1651 Third Ave.
New York, N.Y. 10028
Tel: (212) 860-0300
Telex: 640693 (LBSOFT NYK)
TWX: 710-581-2524

Lifeboat Associates GmbH
Hinterbergstrasse
Postfach 251
6330 Cham, Switzerland
Tel: 042-36-8686
Telex: 865265 (MICO CH)

Lifeboat Inc.
OK Bldg., 5F
1-2-8, Shiba-Daimon
Minato-ku, Tokyo 105, Japan
Tel: 03-437-3901
Telex: 2423296 (LBJTYO)

Intersoft GmbH
Schlossgartenweg 5
D-8045 Ismaning, W. Germany
Tel: 089-966-444
Telex: 5213643 (ISOFD)

Lifeboat Associates, Ltd.
PO Box 125
London WC2H 9LU, England
Tel: 01-836-9028
Telex: 893709 (LBSOFTG)

Lifeboat Associates, SARL
10, Grande Rue Charles de Gaulle
92600 Asnieres, France
Tel: 1-733-08-04
Telex: 250303 (PUBLIC X PARIS)

#1 Lifeboat Associates
World's foremost software source

Mail coupon to: Lifeboat Associates,
1651 Third Avenue, New York, New York 10028
or call (212) 860-0300.

□ Please send me more information on MicroSpell.
□ Please send me a free Lifeboat catalog.

Name____________________ Title____________

Company______________________________

Street________________________________

City______________ State________ Zip________

#1 SOFTWARE WITH FULL SUPPORT

MicroSpell is a trademark of Bob Lucas.
CP/M is a registered trademark of Digital Research.
Copyright © 1981, by Lifeboat Associates

6601

PL/I-80 Review, continued...

versions of the compiler. Some users have had difficulty with debugging their programs as a result. The fourth manual is the LINK manual which contains the addenda to the MAC manual introducing RMAC, a relocating version of the MAC macro-assembler. The librarian LIB is also documented in this manual.

The LINK manual contains important information as to how the various data items are stored in memory. In particular, the manual details the subroutine calls used by the various conversion routines and programmed operators and the input-output complex. This manual is heavy reading and is not recommended for the user who desires only to use the language itself without interface to other routines either in RMAC or other languages. Other sections of the link manual explain the operation of the relocatable loader and library manager. File formats of the relocatable and IRL images are also detailed in this manual.

> *I feel that PL/I-80 is the first true effort at providing the microcomputer community with a decent programming language.*

The MAC manual is the last manual in the set. It is the standard macro assembler language manual and applications guide. In my estimation this is the finest text on 8080/Z80 macro assemblers. No other manual contains such a wide variety of information and depth of coverage as does the MAC manual. Thus, you must mentally merge the RMAC enhancements in the LINK manual with the MAC manual. If you know MAC, you will learn RMAC in about one hour, since nothing is changed by adding RMAC to MAC.

In my addenda there was a treatise on the use of the "picture" clause in commercial processing and some information about the PL/I library license. Hidden in the packet was a PL/I language bibliography with extensive documentation on each book. I recommend the following two texts to all PL/I users:

Augenstein, M. and Tenenbaum, A. *Data Structure and PL/I Programming*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

Huges, Joan K. *PL/I Structured Programming*. New York: John Wiley and Sons, Inc., 1979.

While both texts purport to teach structured programming, I feel that the best structured programming text is:

Kernighan, B. and Ritchie, D: *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

Notice that the words "Structured Programming" do not appear in the title, but that is the essential focus of the book.

Often we measure the quality of a compiler by the code that it generates. It should be noted that the compilers of today spend a large amount of their time in the generation of object code. If the user will tolerate longer compilation times, even better code can be produced, mainly because the state of the art has advanced out of the dark ages of 'fly-by-the-seat-of-your-pants parsing" and "black-magic code generation." Parsing, which back in the days of the first Fortran compilers took up to 40% of the compilation time, now accounts for only about 10%. No doubt about it, PL/I-80 generates good code. It is efficient. One popular Fortran compiler seems to generate a CALL for every statement which appears. Not PL/I-80—it generates a high percentage of in-line program, using CALL only in those situations demanding a non-available operation on the object machine. This results in a more efficient object program. I used both Fortran-80 and PL/I to form the julian date of 200 calendar dates. The PL/I-80 compiler object code was about 30% faster in execution; the resulting program was about 20% smaller than the Fortran-80 program. The question as to what is the best way to measure compiler performance is one with many answers, because one must always qualify the aspect of performance to be measured.

If one is measuring the object code efficiency in a program which will be run hundreds or thousands of times, I suppose it is meaningful. If one is measuring the ease in programming, I believe it means more than all of the other attributes. After all is said and done, today's computers are cheap. Otherwise you and I wouldn't have a roomful of silicon chips with the power of the giant machines of a dozen years ago. What is expensive now is human time. Therefore, the length of time spent programming an application is where the cost is concentrated—not in the time it takes for the application to run on the machine. If we are playing with our machines as a hobby, then the time doesn't mean very much. However, if we are trying to minimize the cost of the machines to maximize profits, then we want to extract the best performance with the cheapest labor investment. This is where PL/I-80 is so valuable. It can turn out good, quality code in a hurry. It is easier to write a PL/I program than to write in Basic. The structured style of PL/I seems to make the program easier to write and debug.

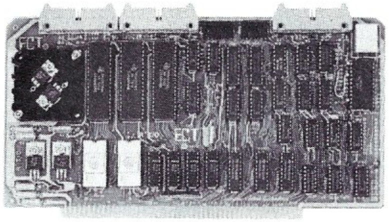
It should be pointed out that Digital Research's PL/I-80 implements the new American National Standards (ANS) Subset-G language, defined especially for minicomputers. It includes all the necessary features of the full PL/I standard, while eliminating useless and redundant forms.

If all of this sounds like a sales pitch for PL/I-80, I cannot help it. I feel that PL/I-80 is the first true effort at providing the microcomputer community with a decent programming language. I do not denegrate the other languages which have been made available. Certain languages such as Basic helped to spur the revolution of personal computers, but an ambitious project such as PL/I-80 has opened a new vista of programming. So, if my "pitch" for PL/I-80 is wildly enthusiastic, it is for good reason. Now, we will be squeezing even more performance out of these micros, and we will be taking a step forward.

To test PL/I-80 I used a 60Kbyte Z-80 system (a minimum of 48Kbytes is required) running at 2MHz. The disks were Micromation Memorex Drives running with a Micromation Doubler and using CP/M 2.2. I used Micromation's CBIOS and "c-2" EPROM. All disk operations were in double density. Compilations always included a storage map and program list. Compile speed ran between 400 to 500 lines per minute. A program to calculate and print the Fibbonaci series using fixed binary (15) declarations took about 1Kbyte of memory for the object code and about 7Kbytes for the run-time subroutines. Calculating

CUSTOM PRODUCTS

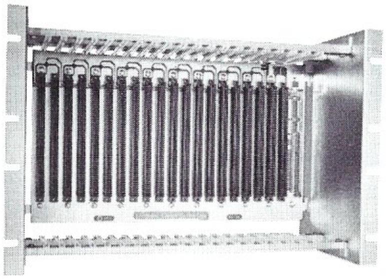
DESIGN • LAYOUT MANUFACTURING



R ²I/O

ROM/RAM & I/O

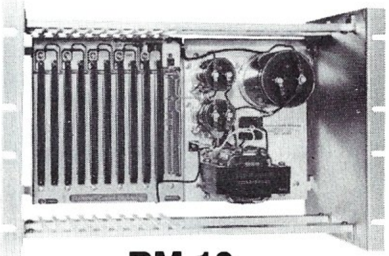
S-100 PRODUCTS



ECT-100-F

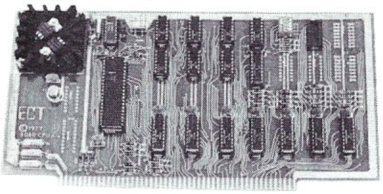
RACKMOUNT CARD CAGES

ECT™



RM-10

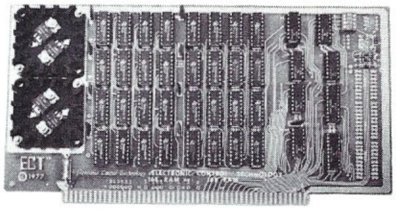
CARD CAGE & POWER SUPPLY



8080 CPU

CENTRAL PROCESSING UNITS

ECT™

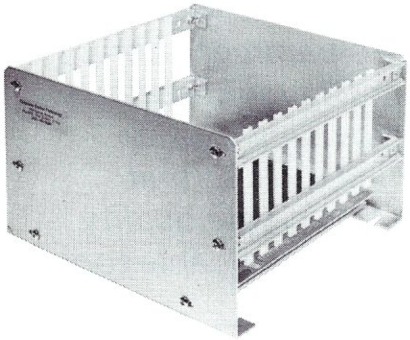


16K RAM

FULLY STATIC MEMORY

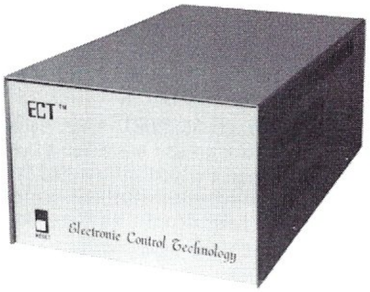
BUILDING BLOCKS FOR MICROCOMPUTER SYSTEMS, DEDICATED CONTROLLERS AND TEST EQUIPMENT

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



CCMB-10-F MIN

6,10 OR 20 SLOT CARD CAGES



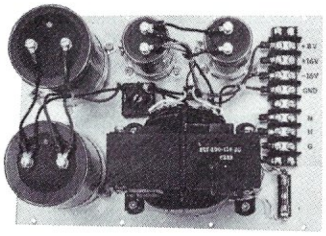
TT-10

TABLE TOP MAINFRAMES

ELECTRONIC CONTROL TECHNOLOGY, INC.

763 Ramsey Ave., Hillside, NJ 07205 (201) 686-8080

SPECIALIZING IN QUALITY MICRO COMPUTER HARDWARE



PS-30 A

POWER SUPPLIES

MULTIBUS® PRODUCTS

MULTIBUS IS A TRADEMARK OF INTEL CORP.

PL/I-80 Review, continued...

FIBB(15) took 3.5 minutes—not bad considering that the calculation was done by means of recursion and dynamic storage allocation.

After the compiler produces the reloctable object code, it must be linkage-edited to form a program. During this phase, routines from the supplied PL/I-80 library (containing over 300 individual routines) and the user library, if appropriate, are added to the PL/I-80 program to form an absolute module. Unfortunately, the LINK program is very slow and takes longer to collect the programs than the original compilation. This should get some attention. You can provide for some increase in speed once you have a PL/I program which will compile by putting your commands to compile PL/I and link PL/I in a submit file. My file also calls for test execution:

```
PLI $1 $$SL
LINK $1
$1
```

which avoids having to type in the stuff for each call.

Since the error checking in PL/I-80 is quite comprehensive, you will have little need to use DDT or similar programs. PL/I-80 gives an extensive set of compilation error messages and execution error messages. You can also use the ON condition clause of PL/I to trap certain execution time errors. The run-time walk back is somewhat difficult to follow, but generally unnecessary since the error message and point at which it occurs usually supplies enough information to spot the offending statement. To debug with DDT you will need to know the internals of PL/I-80. You can get that from the LINK manual.

PL/I-80 represents a valuable addition to any serious programmer's library. In keeping with Digital Research's policy of providing quality programming tools so that programmers can create quality software, PL/I-80 is a valuable tool.

In summary, PL/I-80 represents a radical departure from the current languages available for the 8080/Z80 systems. Digital Research has made a significant contribution to the industry by bringing out PL/I for use on CP/M systems. Previous experience with Digital Research suggests that they will continue to provide a high level of support for this software in keeping with their reputation of reliable support for their customers. In all aspects, PL/I-80 represents a valuable addition to any serious programmer's library. In keeping with Digital Research's policy of providing quality programming tools so that programmers can create quality software, PL/I-80 is a valuable tool.

The complete PL/I-80 package is priced at $500 for non-commercial users. It may be purchased from Digital Research, 801 Lighthouse Ave., Pacific Grove, CA 93950; (408)649-3896 or from any of its dealers. ■

The Best Boards SD Prices Slashed !!!

Single User System

SBC-200, 64K ExpandoRAM II, Versafloppy II, CP/M 2.2

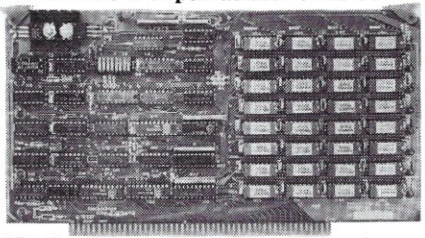
$995.00

4 MHz Z-80A CPU, 64K RAM, serial I/O port, parallel I/O port, double-density disk controller, CP/M 2.2 disk and manuals, system monitor, control and diagnostic software.

-All boards are assembled and tested-

ExpandoRAM III

64K to 256K expandable RAM board



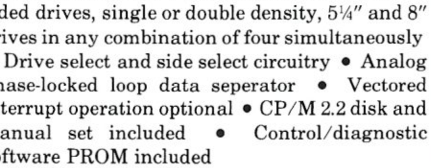
SD Systems has duplicated the famous reliability of their ExpandoRAM I and II boards in the new ExpandoRAM III, a board capable of containing 256K of high speed RAM. Utilizing the new 64K x 1 dymanic RAM chips, you can configure a memory of 64K, 128K, 192K, or 256K, all on one S-100 board. Memory address decoding is done by a programmed bipolar ROM so that the memory map may be dip-switch configured to work with either COSMOS/MPM-type systems or with OASIS-type systems.

Extensive application notes concerning how to operate the ExpandoRAM III with Cromemco, Intersystems, and other popular 4 MHz Z-80 systems are contained in the manual.

MEM-65064A	*64K A & T*	**$495.00**
MEM-65128A	*128K A & T*	**$639.95**
MEM-65192A	*192K A & T*	**$769.95**
MEM-65256A	*256K A & T*	**$879.95**

Versafloppy II

Double density controller with CP/M 2.2



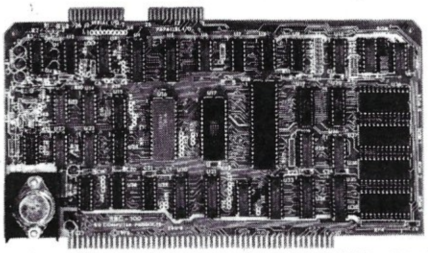
• S-100 bus compatible • IBM 3740 compatible soft sectored format • Controls single and double-sided drives, single or double density, 5¼" and 8" drives in any combination of four simultaneously • Drive select and side select circuitry • Analog phase-locked loop data seperator • Vectored interrupt operation optional • CP/M 2.2 disk and manual set included • Control/diagnostic software PROM included

The Versafloppy II is faster, more stable and more tolerant of bit shift and "jitter" than most controllers. CP/M 2.2 and all necessary control and diagnostic software are included.

IOD-1160A *A & T with CP/M 2.2* .. **$370.00**

SBC-200

2 or 4 MHz single board computer



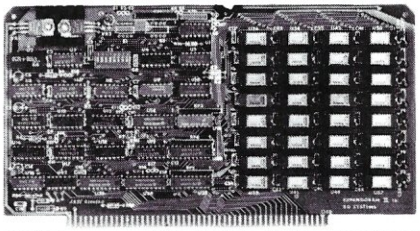
• S-100 bus compatible • Powerful 4MHz Z-80A CPU • Synchronous/asynchronous serial I/O port with RS-232 interface and software programmable baud rates up to 9600 baud • Parallel input and parallel output port • Four channel counter/timer • Four maskable, vectored interrupt inputs and a non-maskable interrupt • 1K of on-board RAM • Up to 32K of on-board ROM • System monitor PROM included

The SBC-200 is an excellent CPU board to base a microcomputer system around. With on-board RAM, ROM, and I/O, the SBC-200 allows you to build a powerful three-board system that has the same features found in most five-board microcomputers. The SBC-200 is compatible with both single-user and multi-user systems.

CPU-30200A *A & T with monitor* . **$299.95**

ExpandoRAM II

16K to 64K expandable RAM board



• S-100 bus compatible • Up to 4MHz operation • Expandable from 16K to 64K • Uses 16 x 1 4116 memory chips • Page mode operation allows up to 8 memory boards on the bus • Phantom output disable • Invisible on-board refresh

The ExpandoRAM II is compatible with most S-100 CPUs. When other SD System' series II boards are combined with the ExpandoRAM II, they create a microcomputer system with exceptional capabilities and features.

MEM-16630A	*16K A & T*	**$325.00**
MEM-32631A	*32K A & T*	**$345.00**
MEM-48632A	*48K A & T*	**$365.00**
MEM-64633A	*64K A & T*	**$385.00**

COSMOS

Multi-user operating system

• Multi-user disk operating system • Allows up to 8 users to run independent jobs concurrently • Each user has a seperate file directory

COMOS supports all the file structures of CP/M 2.2, and is compatible at the applications program level with CP/M 2.2, so that most programs written to run under CP/M 2.2 or SDOS will also run under COSMOS.

SFC-55009039F *COSMOS on 8" disk* **$395.00**

Multi-User System

SBC-200, 256K ExpandoRAM III, Versafloppy II, MPC-4 COSMOS Multi-User Operating System, C BASIC II

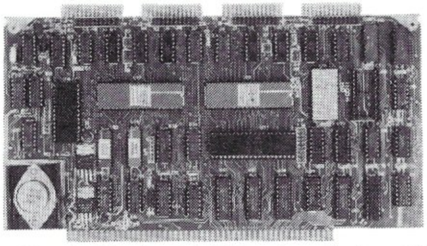
$1995.00

Two Z-80A CPUs (4 MHz), 256K RAM, 5 serial I/O ports with independently programmable baud rates and vectored interrupts, parallel input port, parallel output port, 8 counter/timer channels, real time clock, single and double sided/single or double density disk controller for 5¼" and 8" drives, up to 36K of on-board ROM, CP/M 2.2 compatible COSMOS interrupt driven multi-user disk operating system, allows up to 8 users to run independent jobs concurrently, C BASIC II, control and diagnostic software in PROM included.

-All boards are assembled and tested-

MPC-4

Intelligent communications interface



• Four buffered serial I/O ports • On-board Z-80A processor • Four CTC channels • Independently programmable baud rates • Vectored interrupt capability • Up to 4K of on-board PROM • Up to 2K of on-board RAM • On-board firmware

This is not just another four-port serial I/O board! The on-board processor and firmware provide sufficient intelligence to allow the MPC-4 to handle time consuming I/O tasks, rather than loading down your CPU. To increase overall efficiency, each serial channel has an 80 character input buffer and a 128 character output buffer. The on-board firmware can be modified to make the board SDLC or BISYNC compatible. In combination with SD's COSMOS operating system (which is included with the MPC-4), this board makes a perfect building block for a multi-user system.

IOI-1504A *A & T with COSMOS* .. **$495.00**

Place Orders Toll Free

Continental U.S.	*Inside California*
800-421-5500	**800-262-1710**

For Technical Inquires or Customer Service call:

213-973-7707

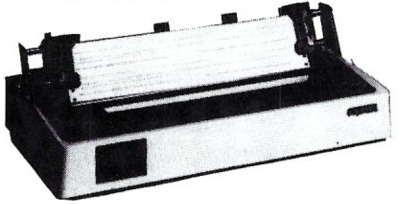
JADE
Computer Products

4901 W. Rosecrans, Hawthorne, Ca 90250

TERMS of SALE: Cash, checks, credit cards, or Purchase Orders from qualified firms and institutions. **Minimum Order $15.00.** California residents add 6% tax. Minimum shipping & handling charge $3.00. **Pricing & availibility subject to change**

JADE
Computer Products

Printers



BETTER THAN EPSON ! - Okidata

Microline 82A *80/132 column,* ***120 CPS****, 9 x 9 dot matrix, friction feed, pin feed, adjustable tractor feed (removable), handles 4 part forms up to 9.5" wide, rear & bottom feed, paper tear bar, 100% duty cycle/200,000,000 character print head, bi-directional/logic seeking, both serial & parallel interfaces included, front panel switch & program control of 10 different form lengths, uses inexpensive spool type ribbons, double width & condensed characters, true lower case descenders & graphics*

PRM-43082	*with FREE tractor*	**$539.95**

Microline 83A *132/232 column, 120 CPS, handles forms up to 15" wide, plus all the features of the 82A.*

PRM-43083	*with FREE tractor*	**$749.95**

PRA-27081A	*Apple card*	**$39.95**
PRA-27082A	*Apple cable*	**$19.95**
PRA-27087A	*TRS-80 cable*	**$24.95**
PRA-43080	*Extra ribbons pkg. of 2*	**$9.95**

INEXPENSIVE PRINTERS - Epson

MX-70 *80 column, 80 CPS, 5 x 7 dot matrix, adjustable tractor feed, & graphics*

PRM-27070	*List $459*	**$399.95**

MX-80 *80 column, 80 CPS, bi-directional/logic seeking printing, 9 x 9 dot matrix, adjustable tractor feed, & 64 graphics characters*

PRM-27080	*List $645*	**$469.95**

MX-80FT *same as MX-80 with friction feed added.*

PRM-27082	*List $745*	**$559.95**

MX-100 *132 column, correspondence quality, graphics, up to 15" paper, friction feed & adjustable tractor feed, 9 x 9 dot matrix, 80 CPS.*

PRM-27100	*List $945*	**$759.95**

PRA-27084	*Serial interface*	**$69.95**
PRA-27088	*Serial intf & 2K buffer*	**$144.95**
PRA-27081	*Apple card*	**$74.95**
PRA-27082	*Apple cable*	**$22.95**
PRA-27086	*IEEE 488 card*	**$52.95**
PRA-27087	*TRS-80 cable*	**$32.95**
PRA-27085	*Graftrax II*	**$95.00**
PRA-27083	*Extra ribbon*	**$14.95**

NEC 7700 & 3500

NEC Spinwriter w/Intelligent Controller

Standard serial, Centronics parallel, and current loop interfaces • Selectable baud rates 50 to 19,200 • Automatic bidirectional printing • Logic seeking • 650 character buffer with optional 16K buffer • 55 characters per second print speed • Comes with vertical forms tractor, ribbon, thimble and cable • Diablo compatible software • Available with or without optional front panel

PRD-55511	*1K no front panel*	**$2795.00**
PRD-55512	*16K no front panel*	**$2895.00**
PRD-55515	*1K w/front panel*	**$2995.00**
PRD-55516	*16K w/front panel*	**$3095.00**

Intersell NEC 3500Q

New from NEC - the 3500 series Spinwriters. Incorporates all the features and reliability of the 5500 and 7700 series Spinwriters into an inexpensive 30 CPS letter quality printer with an optional bi-directional tractor assembly.

PRD-55351	*3500Q 1K*	**$1995.00**
PRD-55352	*3500Q 16K*	**$2095.00**
PRA-55100	*Deluxe tractor option*	**$300.00**

Accessories for Apple

16K MEMORY UPGRADE

Add 16K of RAM to your TRS-80, Apple, or Exidy in just minutes. We've sold thousands of these 16K RAM upgrades which include the appropriate memory chips (as specified by the manufacturer), all necessary jumper blocks, fool-proof instructions, and our 1 year guarantee.

MEX-16100K	*TRS-80 kit*	**$25.00**
MEX-16101K	*Apple kit*	**$25.00**
MEX-16102K	*Exidy kit*	**$25.00**

16K RAM CARD - for Apple II

Expand your Apple to 64K, 1 year warranty

MEX-16500A	*Save $70.00 !!!*	**$129.95**

Z-80* CARD for APPLE

Two computers in one, Z-80 & 6502, more than doubles the power & potential of your Apple, includes Z-80 CPU card, CP/M 2.2, & BASIC-80*

CPX-30800A	*A & T*	**$299.95**

8" DISK CONTROLLER

New from Vista Computer, single or double sided, single or double density, compatible with DOS 3.2/3.3, Pascal, & CPM 2.2, Shugart & Qume compatible

IOD-2700A	*A & T*	**$499.95**

2 MEGABYTES for Apple II

Complete package includes: Two 8" double-density disk drives, Vista double-density 8" disk controller, cabinet, power supply, & cables, DOS 3.2/3.3, CP/M 2.2, & Pascal compatible.

1 MegaByte Package *(Kit)*	**$1495.00**
1 MegaByte Package *(A & T)*	**$1695.00**
2 MegaByte Package *(Kit)*	**$1795.00**
2 MegaByte Package *(A & T)*	**$19.95**

CPS MULTICARD - Mtn. Computer

Three cards in one! Real time clock/calendar, serial interface, & parallel interface - all on one card.

IOX-2300A	*A & T*	**$199.95**

AIO, ASIO, APIO - S.S.M.

Parallel & serial interface for your Apple (see Byte pg 11)

IOI-2050K	*Par & Ser kit*	**$139.95**
IOI-2050A	*Par & Ser A & T*	**$169.95**
IOI-2052K	*Serial kit*	**$89.95**
IOI-2052A	*Serial A & T*	**$99.95**
IOI-2054K	*Parallel kit*	**$69.95**
IOI-2054A	*Parallel A & T*	**$89.95**

A488 - S.S.M.

IEEE 488 controller, uses simple basic commands, includes firmware and cable, 1 year guarantee, (see April Byte pg 11)

IOX-7488A	*A & T*	**$399.95**

Modems

CAT MODEMS - Novation

CAT *300 baud, acoustic, answer/orginate*

IOM-5200A	*List $189.95*	**$149.95**

D-CAT *300 baud direct connect, answer/orginate*

IOM-5201A	*List $199.95*	**$169.95**

AUTO-CAT *Auto answer/orginate, direct connect*

IOM-5230A	*List $299.95*	**$239.95**

Apple-CAT - Novation

Software selectable 1200 or 300 baud, direct connect, auto-answer/auto-dial, auxiliary 3-wire RS232C serial port for printer.

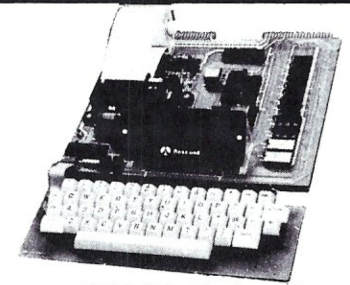
IOM-5232A	*Save $50.00!!!*	**$325.00**

SMARTMODEM - Hayes

Sophisticated direct-connect auto-answer/auto-dial modem, touch-tone or pulse dialing, RS-232C interface, programmable

IOM-5400A	*Smartmodem*	**$269.95**

Single Board Computer



AIM-65 - Rockwell

6502 computer with alphanumeric display, printer, & keyboard, and complete instructional manuals

CPK-50165	*1K AIM*	**$424.95**
CPK-50465	*4K AIM*	**$474.95**
SFK-74600008E	*8K BASIC ROM*	**$64.95**
SFK-64600004E	*4K assembler ROM*	**$43.95**
PSX-030A	*Power supply*	**$64.95**
ENX-000002	*Enclosure*	**$54.95**

4K AIM, 8K BASIC, power supply, & enclosure

Special package price	**$649.95**

Z-80 STARTER KIT - SD Systems

Complete Z-80 microcomputer with RAM, ROM, I/O, keyboard, display, kludge area, manual, & workbook

CPS-30100K	*KIT*	**$299.95**
CPS-30100A	*A & T*	**$469.95**

SYM-1 - Synertek Systems

Single board computer with 1K of RAM, 4K of ROM, key-pad, LED display, 20ma & cassette interface on board.

CPK-50020A	*A & T*	**$249.95**

Video Monitors

HI-RES 12" GREEN - Zenith

15 MHz bandwidth, 700 lines/inch, P31 green phosphor, switchable 40 or 80 columns, small, light-weight & portable.

VDM-201201	*List price $150.00*	**$118.95**

Leedex / Amdek

Reasonably priced video monitors

VDM-801210	*Video 100 12" B&W*	**$139.95**
VDM-801230	*Video 100-80 12" B& W*	**$179.95**
VDM-801250	*12" Green Phospor*	**$169.95**
VDC-801310	*13" Color I*	**$379.95**

12" COLOR MONITOR - NEC

Hi-res monitor with audio & sculptured case

VDC-651212	*Color Monitor*	**$479.95**

12" GREEN SCREEN - NEC

20 MHz, P31 phosphor video monitor with audio, exceptionally high resolution - A fantastic monitor at a very reasonable price

VDM-651200	*Special Sale Price*	**$199.95**

Video Terminals

AMBER SCREEN - Volker Craig

Detachable keyboard, amber on black display, 7 x 9 dot matrix, 10 program function keys, 14 key numeric pad, 12" non-glare screen, 50 to 19,200 baud, direct cursor control, auxiliary bi-directional serial port

VDT-351200	*List $795.00*	**$645.00**

VIEWPIONT - ADDS

Detachable keyboard, serial RS232C interface, baud rates from 110 to 19,200, auxiliary serial output port, 24 x 80 display,

VDT-501210	*Sale Priced*	**$639.95**

TELEVIDEO 950

VDT-901250	*List $1195.00*	**$995.00**

DIALOGUE 80 - Ampex

VDT-230080	*List $1195.00*	**$895.00**

JADE
Computer Products

S-100 CPU Boards

THE BIG Z* - Jade

2 or 4 MHz switchable Z-80 CPU with serial I/O, accomodates 2708, 2716, or 2732 EPROM, baud rates from 75 to 9600*

CPU-30201K	*Kit*	$139.95
CPU-30201A	*A & T*	$189.95
CPU-30200B	*Bare board*	$35.00

2810 Z-80* CPU - Cal Comp Sys

2/4 MHz Z-80A CPU with RS-232C serial I/O port and on-board MOSS 2.2 monitor PROM, front panel compatible.*

CPU-30400A	*A & T*	$269.95

CB-2 Z-80 CPU - S.S.M.

2 or 4 MHz Z-80 CPU board with provision for up to 8K of ROM or 4K of RAM on board, extended addressing, IEEE S-100, front panel compatible.

CPU-30300K	*Kit*	$239.95
CPU-30300A	*A & T*	$299.95

S-100 PROM Boards

PROM-100 - SD Systems

2708, 2716, 2732 EPROM programmer w/software

MEM-99520K	*Kit*	$189.95
MEM-99520A	*A & T*	$249.95

PB-1 - S.S.M.

2708, 2716 EPROM board with built-in programmer

MEM-99510K	*Kit*	$154.95
MEM-99510A	*A & T*	$219.95

EPROM BOARD - Jade

16K or 32K uses 2708's or 2716's, 1K boundary

MEM-16230K	*Kit*	$79.95
MEM-16230A	*A & T*	$119.95

S-100 Video Boards

VB-3 - S.S.M.

80 characters x 24 lines expandable to 80 x 48 for a full page of text, upper & lower case, 256 user defined symbols, 160 x 192 graphics matrix, memory mapped, has key board input.

IOV-1095K	*4 MHz kit*	$349.95
IOV-1095A	*4 MHz A & T*	$439.95
IOV-1096K	*80 x 48 upgrade*	$39.95

VDB-8024 - SD Systems

80 x 24 I/O mapped video board with keyboard I/O, and on-board Z-80A.*

IOV-1020A	*A & T*	$459.95

VIDEO BOARD - S.S.M.

64 characters x 16 lines, 128 x 48 matrix for graphics, full upper/lower case ASCII character set, numbers, symbols, and greek letters, normal/reverse/blinking video, S-100.

IOV-1051K	*Kit*	$149.95
IOV-1051A	*A & T*	$219.95
IOV-1051B	*Bare board*	$34.95

S-100 Motherboards

ISO-BUS - Jade

Silent, simple, and on sale - a better motherboard

6 Slot (5¼" x 8⅝")

MBS-061B	*Bare board*	$19.95
MBS-061K	*Kit*	$39.95
MBS-061A	*A & T*	$49.95

12 Slot (9¾" x 8⅝")

MBS-121B	*Bare board*	$29.95
MBS-121K	*Kit*	$69.95
MBS-121A	*A & T*	$89.95

18 Slot (14½" x 8⅝")

MBS-181B	*Bare board*	$49.95
MBS-181K	*Kit*	$99.95
MBS-181A	*A & T*	$139.95

S-100 RAM Boards

MEMORY BANK - Jade

4 MHz, S-100, bank selectable, expandable from 16K to 64K

MEM-99730B	*Bare Board*	$49.95
MEM-99730K	*Kit no RAM*	$199.95
MEM-32731K	*32K Kit*	$239.95
MEM-64733K	*64K Kit*	$279.95
Assembled & Tested		add $50.00

64K RAM - Calif Computer Sys

4 MHz bank port / bank byte selectable, extended addressing, 16K bank selectable, PHANTOM line allows memory overlay, 8080 / Z-80 / front panel compatible.

MEM-64565A	*A & T*	$575.00

64K STATIC RAM - Mem Merchant

64K static S-100 RAM card, 4-16K banks, up to 8MHz

MEM-64400A	*A & T*	$789.95

32K STATIC RAM - Jade

2 or 4 MHz expandable static RAM board uses 2114L's

MEM-16151K	*16K 4 MHz kit*	$169.95
MEM-32151K	*32K 4 MHz kit*	$299.95
Assembled & tested		add $50.00

16K STATIC RAM - Mem Merchant

4 MHz 16K static RAM board, IEEE S-100, bank selectable, Phantom capability, addressable in 4K blocks, "disable-able" in 1K segments, extended addressing, low power

MEM-16171A	*A & T*	$164.95

S-100 Disk Controllers

DOUBLE-D - Jade

Double density controller with the inside track, on-board Z-80A, printer port, IEEE S-100, can function on an interrupt driven buss*

IOD-1200K	*Kit*	$299.95
IOD-1200A	*A & T*	$375.00
IOD-1200B	*Bare board*	$59.95

DOUBLE DENSITY - Cal Comp Sys

5¼" and 8" disk controller, single or double density, with on-board boot loader ROM, and free CP/M 2.2 and manual set.*

IOD-1300A	*A & T*	$374.95

S-100 I/O Boards

S.P.I.C. - Jade

Our new I/O card with 2 SIO's, 4 CTC's, and 1 PIO

IOI-1045K	*2 CTC's, 1 SIO, 1 PIO*	$179.95
IOI-1045A	*A & T*	$239.95
IOI-1046K	*4 CTC's, 2 SIO's, 1 PIO*	$219.95
IOI-1046A	*A & T*	$299.95
IOI-1045B	*Bare board w/ manual*	$49.95

I/O-4 - S.S.M.

2 serial I/O ports plus 2 parallel I/O ports

IOI-1010K	*Kit*	$179.95
IOI-1010A	*A & T*	$249.95
IOI-1010B	*Bare board*	$35.00

S-100 Mainframes

MAINFRAME - Cal Comp Sys

12 slot S-100 mainframe with 20 amp power supply

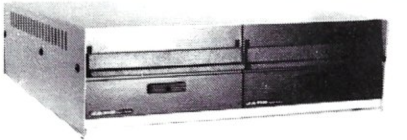
ENC-112105	*Kit*	$329.95
ENC-112106	*A & T*	$399.95

DISK MAINFRAME - N.P.C.

Holds 2 8" drives and a 12 slot S-100 system. Attractive metal cabinet with 12 slot motherboard & card cage, power supply, dual fans, lighted switch, and other professional features

ENS-112325	*with 25 amp p.s.*	$699.95

Disk Drives



Handsome metal cabinet with proportionally balanced air flow system • Rugged dual drive power supply • Power cable kit • Power switch, line cord, fuse holder, cooling fan • Never-Mar rubber feet • All necessary hardware to mount 2-8" disk drives, power supply, and fan • Does not include signal cable

Dual 8" Subassembly Cabinet

END-000420	*Bare cabinet*	$59.95
END-000421	*Cabinet kit*	$225.00
END-000431	*A & T*	$359.95

8" Disk Drive Subsystems
Single Sided, Double Density

END-000423	*Kit w/2 FD100-8Ds*	$924.95
END-000424	*A & T w/2 FD100-8Ds*	$1124.95
END-000433	*Kit w/2 SA-801Rs*	$999.95
END-000434	*A & T w/2 SA-801Rs*	$1195.00

8" Disk Drive Subsystems
Double Sided, Double Density

END-000426	*Kit w/2 DT-8s*	$1224.95
END-000427	*A & T w/2 DT-8s*	$1424.95
END-000436	*Kit w/2 SA-851Rs*	$1495.00
END-000437	*A & T w/2 SA-851Rs*	$1695.00

QUME DT-8

8" Double-Sided, Double-Density Disk Drive

1 Drive	$524.95 each
2 Drives	$499.95 each
10 Drives	$479.95 each

Jade Part Number MSF-750080

Shugart 801R

8" Single-Sided, Double-Density Disk Drive

1 Drive	$394.95 each
2 Drives	$389.95 each

Jade Part Number MSF-10801R

SIEMENS 8"

8" Single-Sided, Double-Density Disk Drive

1 Drive	$384.95 each
2 Drives	$349.95 each
10 Drives	$324.95 each

Jade Part Number MSF-201120

MPI B-51

5¼" Single-Sided, Double-Density Disk Drive

1 Drive	$234.95 each
2 Drives	$224.95 each
10 Drives	$219.95 each

Jade Part Number MSM-155100

END-000213	*Case & power supply*	$74.95

An Introduction to the C Programming Language (Part II)

by David A. Gewirtz

In this, the second in a two-part series, the author evaluates C compiler implementations.

There are a number of things to consider when comparing different implementations of a single language. Usually the most efficient way to evaluate what is best for a particular purpose is to look at all of them together.

In any computer-related operation, speed considerations are important, so one thing to check is the execution speed of programs. Additionally, to anyone who will be using the compiler often, speed of compilation is very important. No one likes waiting hours to see the results of the latest program modification.

Since implementations of a language vary, it is very important to see how close an implementation is to the "standard" language specification. It could be a near match, but leave out some important features. An analogy can be seen in the S-100 bus. A memory board may be "close" to the standard, but it wouldn't be of much use if the manufacturer just happened to leave out the fifth address line. Similarly, many features of a language can be left out without ill effect, but most key features should be included.

Finally, cost and system size are very important. You may not have a need for a very expensive compiler or may not be able to afford one. If you only have 32K bytes of memory in your system, a compiler that requires a minimum of 56K will be of little use unless you upgrade. Somewhat related to system size is the size of a compiled program. It's important to know just how much overhead each completed program has to lug around to work properly.

In order to compare the C compilers reviewed here, several tests were made. The results are shown in the charts and tables in this article. However, they require a brief description to actually understand them.

First, there is the problem of testing execution speed of the code generated by the compiler. Many benchmark tests run a series of programs through loops that repeat a number of different numerical and floating point calculations. This is not good for a systems language such as C. The six programs used in this performance evaluation (PE) test most of the features of C in such a way as to gain a good understanding of each compiler's internal operation. Each program loops through a set of simple operations that tests that particular feature. The first program (PE1) is a simple counting loop with no operations inside the loops. Since the tiny-c and Small-C compilers have not implemented "for" loops, the tests for those compilers use the "while" structure. BDS C and Whitesmiths C do use the "for" structure. The next test, PE2, performs integer calculations inside a simple counting structure and tests how fast each compiler can perform the mathematical functions of addition, subtraction, multiplication and division. PE3 tests the execution speed of "if/then" statements. It's important to see how fast a compiler can evaluate a conditional expression and follow a path. To keep everything consistent, each path does the same thing, if taken. Since a large portion of C programs make extensive use of pointers and indirection, this is another very important thing to test in PE4. Finally, C

Small-C is great as an inexpensive alternative to assembler and for the person who wants to experiment with an inexpensive compiler.

David A. Gewirtz, 38-67 Taylor Rd., Fairlawn, NJ 07410.

programs are very block-structured, and use functions extensively. The final two tests examine the speed at which functions are called, both with (PE6) and without (PE5) argument passing. In order to be sure of the integrity of the run-time measurements, three measurements were taken from three runs of each program for each compiler. The results were taken from three runs of each program for each compiler, then averaged together to come up with the final run time listed. All tests were made with a digital stop watch. All of the tests were done on a 60K byte double density disk 8" disk system using a Z80-A microprocessor running at a 4MHz clock speed.

The Whitesmiths compiler is useful mostly to someone who is designing large, portable systems. A program written in Whitesmiths C on CP/M is portable to the VAX, PDP-11, LSI-11, and 68000. It also contains the entire C syntax.

Generally, both the BDS and Whitesmiths C compilers execute programs at about the same speed. Whitesmiths is faster at simple counting, conditional evaluation, and indirection. BDS is faster at integer calculation and function calling, both with and without arguments. The most significant difference is in the area of the integer calculations. While the integer calculation test on the BDS takes about one third as long as the counting loop, the same test compiled with the Whitesmiths takes longer than the counting loop. Although untested, this would imply that floating point calculations might also be rather slow. These tests tie BDS C and Whitesmiths C for the first place position in the execution speed tests. It's interesting to note that the Whitesmiths C compiler is written in C, while BDS C is written in 8080 assembler code.

The $15 Small-C compiler is the runner up in the speed tests. It is about one-half the speed of BDS and Whitesmiths. For a very inexpensive compiler, this is a real winning point.

Last in the speed trials comes the tiny-c Two compiler. It averages thirty times slower than BDS and Whitesmiths together and twenty-two times slower than the speeds of all of the other three compilers averaged together. Although faster than the tiny-c interpreter, this compiler is not as fast as one would expect it. The longest running test program of the other three compilers (PE4 on Small-C) took 9 minutes, 24 seconds to execute. This same program took two hours and twenty-seven minutes to execute using the tiny-c TWO compiler. This is quite a difference, even without considering the fact that Small-C is $15 and tiny-c TWO is $250.

The next thing tested was the speed of compilation. These tests measured the time it took to go from source code to executable object code, including assembly and linkage if necessary. The fastest was BDS C, with an average compile and link time of 29.7 seconds. This is even faster than the Digital Research MAC Macro Assembler would assemble the code produced by Small-C. The second fastest was tiny-c TWO, pulling up from last place in the execution speed runs to second place with an average 63 second compile/link time. Obviously, they should have the compiler spend more time to produce faster code. Next in line was Small-C. This was interesting to measure as the compilation time was measured from the Small-C compiler. Assembly of the assembler source code produced by the compiler and load time of the hex file produced also had to be measured. Together the whole thing totaled about a three and one-half minute compilation and linkage time. Finally, bringing up the rear is the Whitesmiths C compiler. Whitesmiths takes an average of 246.3 seconds (just over four minutes) to compile and link a program. Most of this time, about three minutes, was spent in the linkage stage. I suspect this is because it has over one hundred and sixty functions that the linker must sift through.

The last type of empirical measurement was final object file size. These measurements were taken by using the CP/M STAT command. The results are formatted in terms of records and K's of bytes. The least amount of space was taken up by tiny-c TWO, with about two records and 2K bytes. The space used by tiny-c TWO is so small because the entire run-time system, usually included with the object code, is included in the separate shell module used to run the programs. Predictably, next in line are Small-C programs. Following that is BDS C and finally, with much larger object code files than all of the others, are the programs generated by Whitesmiths. The size of the object file is usually dependent on how powerful the implementation is and how much support software must be carried along. It does however, seem that the Whitesmiths files are still a bit larger than they need be.

Looking at all of this information, it is very difficult to come up with any definite winners or losers. Each different implementation has its advantages and disadvantages. Whitesmiths is a complete implementation and is as fast as BDS C, but it takes a long time to compile and its purchase price is high. Small-C lacks many features, but is fast and very inexpensive. Tiny-c TWO is slow, but comes with impressive documentation, is a terrific learning tool, and works very well with its interpreter as a development tool.

The BD Software C compiler seems to be with the most universal appeal. At $145, it is a relatively inexpensive, quality compiler. It is fast, easy to use, and fairly complete.

Fortunately each of the four compilers seems to appeal to a certain type of user with only minimal overlap. Small-C is great as an inexpensive alternative to assembler and

Compiler Test Results

BDS C

Average Compilation and Linkage Time: 29.7 seconds
Average Final Program Size: 26 recs 4K bytes

	Run-time (seconds)
PE1 - Simple Counting Loop	22.5
PE2 - Simple Count and Integer Calculation	8.0
PE3 - Conditional Evaluation	7.5
PE4 - Indirectional (Pointer) Operations	256.8
PE5 - Simple Function Calling (no arguments)	38.1
PE6 - Function Calling with Argument Passing	87.7

Small-C

Average Compilation and Linkage Time: 203 seconds
Compilation (C80): 155.5 seconds
Assembling (MAC): 38.2 seconds
Loading (LOAD): 7.8 seconds

Average Final Program Size: 19 recs 4K bytes

	Run-time (seconds)
PE1 - Simple Counting Loop	49.1
PE2 - Simple Count and Integer Calcualtion	41.4
PE3 - Conditional Evaluation	15.2
PE4 - Indirection (Pointer) Operations	564.9
PE5 - Simple Function Calling (no arguments)	55.7
PE6 - Function Calling with Argument Passing	123.0

Tiny-c TWO

Average Compilation and Linkage Time: 63 seconds
Average Final Program Size: 2 recs 2K bytes

	Run-time (seconds)
PE1 - Simple Counting Loop	688.0
PE2 - Simple Count and Integer Calculation	166.0
PE3 - Conditional Evaluation	186.0
PE4 - Indirection (Pointer) Operations	8820.0
PE5 - Simple Function Calling (no arguments)	1168.0
PE6 - Function Calling with Argument Passing	1973.0

Whitesmiths C

Average Compilation and Linkage Time: 246.3 seconds
Average Final Program Size: 123 recs 16K bytes

	Run-time (seconds)
PE1 - Simple Counting Loop	16.9
PE2 - Simple Count and Integer Calculation	17.1
PE3 - Conditional Evaluation	6.2
PE4 - Indirection (Pointer) Operations	221.4
PE5 - Simple Function Calling (no arguments)	47.1
PE6 - Function Calling with Argument Passing	98.3

Functions of the C Language

Function	BDS C	Small-C	Tiny-c TWO	Whitesmiths C
do/while	X			X
for	X			X
goto	X			X
if/else	X	X	X	X
switch/case	X			X
while	X	X	X	X
return	X	X	X	X
break	X	X	X	X
default	X			X

Storage Classes

Storage Class	BDS C	Small-C	Tiny-c TWO	Whitesmiths C
extern	X		X	X
auto	X			X
static			X	X
register				X
typedef				X

Data Types

Data Types	BDS C	Small-C	Tiny-c TWO	Whitesmiths C
char	X	X	X	X
int	X	X	X	X
long			X	X
float				X
double				X
struct	X			X
union	X			X
pointer to	X	X	X	X
array of	X	X	X	X

Unary Operators

Unary Operator	BDS C	Small-C	Tiny-c TWO	Whitesmiths C
*p	X	X	(implicit)	X
&x	X	X	X	X
+x	X		X	X
-x	X	X	X	X
++x	X	X	X	X
--x	X	X	X	X
x++	X	X	X	X
x--	X	X	X	X
~x	X		X	X
!x	X		X	X
(type-name)x				X
sizeof x	X			X
sizeof (type-name)				X
				X

Binary Operators

Binary Operator		BDS C	Small-C	Tiny-c TWO	Whitesmiths C
x*y		X	X	X	X
x/y		X	X	X	X
x%y		X	X	X	X
x+y		X	X	X	X
x-y		X	X	X	X
x<<y		X	X	X	X
x>>y		X	X	X	X
x<y		X	X	X	X
x>y		X	X	X	X
x<=y		X	X	X	X
x>=y		X	X	X	X
x==y		X	X	X	X
x!=y		X	X	X	X
x&y		X	X	X	X
x^y		X	X	X	X
x | y		X	X	X	X
x&&y		X			X
x || y		X			X
t?x:y		X			X
x=y		X	X		X
x*=y		X			X
x/=y		X			X
x%=y		X			X
x+=y		X			X
x-=y	*	X			X
x<<=y		X			X
x>>=y		X			X
x&=y		X			X
x^=y		X			X
x |=y		X			X

*BDS-C and Whitesmiths C use the "OP=" shorthand while Small-C and TIny-C do not. However, these operations can be accomplished in Small-C and Tiny-C in the standard manner.

for the person who wants to experiment with an inexpensive compiler. Since it comes with source code, it can be extensively modified by any "hacker." The Whitesmiths compiler is useful mostly to someone who is designing large, portable systems. A program written in Whitesmiths C on CP/M is portable to the VAX, PDP-11 LSI-11, and 68000. It also contains the entire C syntax. Tiny-c TWO is best for someone who still wants to learn, and also upgrade from an interpreter to a compiler. And, it comes with complete source code and a user-modifiable command processing shell. The BD Software C compiler seems to be the one with the most universal appeal. At $145, it is a relatively inexpensive quality compiler. It is fast, easy to use, and fairly complete. I have been using the compiler for quite some time and have found everything implemented that I really needed, with the possible exception of the static data type.

All of these compilers generated error messages during compilation and linkage. Although they were adequate and accurate, not one would win an award for clarity. Error messages are supposed to give *useful* information about errors to the programmer to help debug programs. Also, it would be nice to have a listing of *all* error messages in the manual with coherent explanations of what the messages mean. The tiny-c manual was closest to this.

While we're critiquing manuals, I would like to see a complete specification of the program, language, or utility on the first page. This description should include the minimum amount of memory needed, the version number, and the address and phone number of the folks to call for help. One last thing that I would like to see with these, and all other higher level compilers on micros, are debugging aids. Big machines have debugging programs that allow tracing through the high level language statements, placing breakpoints, changing values, and so on. Instead of looking in SID (Digital Research's "Symbolic Instruction Debugger") for an 'LDA A,var', it would be nice to have a breakpoint at 'a=var'. The closest to this is BDS, which generates a simple table acceptable to SID.

An interesting thing about these compilers is their quality. Although some of them may be faster or slower than the others, and may be missing some features I would like to see, they are all well-executed products. The compilers are complete and well thought-out. They are accompanied by reasonable documentation, although the documentation from Whitesmiths was an experience.

Finally, I found the customer service people from all of the companies to be very helpful. One minor note is that they did know I was reveiwing their compilers, so I'm not sure how I would have been treated otherwise. I also cannot testify to the quality of customer support at Lifeboat Associates, the distributor of BDS C. I dealt directly with Leor Zolman, the author, who was extremely helpful. One final observation concerns both Tom Gibson of tiny-c associates and Leor Zolman of BD Software. I have spoken with many people who have also dealt with them, and have learned that they have very good reputations. BDS C, Small-c, tiny-c TWO, and Whitesmiths C have all impressed me immensely. ■

BDS C Evaluation Programs

```
/*
                Performance Evaluation Program #1

                     Simple Counting Loops

                            (BDS C)

*/

main()
{
        int i, j;
        printf("Start of Run\n");
        for (i=1; i!=5000; ++i)
        {
                for (j=1; j!=100; ++j)
                {
                }
        }
        printf("End of Run\n");
}
```

```
/*

                Performance Evaluation Program #2

             Simple Count and Integer Calculation

                            (BDS C)

*/

main()
{
        int i, j, k, l;
        printf("Start of Run\n");
        for (i=1; i!=30000; ++i)
        {
                j = (5*607+7)/32;
                k = (j+47)*61;
        }
        printf("End of Run\n");
}
```

```
/*

                Performance Evaluation Program #3

                         Conditionals

                            (BDS C)

*/

main()
{
        int i, j;
        printf("Start of Run\n");
        j = 2500;
        for (i=1; i!=30000; ++i)
        {

                if (i < j)
                {
                        j = 2500;
                }
                else
                {
                        j = 2500;
                }
                if (i > j)
                {
                        j = 2500;
                }
                else
                {
                        j = 2500;
                }
                if (i <= j)
                {
                        j = 2500;
                }
                else
                {
                        j = 2500;
                }
                if (i >= j)
                {
                        j = 2500;
                }
                else
                {
                        j = 2500;
                }
        }
        printf("End of Run\n");
}
```

```
/*

                Peformance Evaluation Program #4

                       Pointer Operations

                            (BDS C)

*/

main()
{
        char arry[128], *ptr;
        int i;
        printf("Start of Run\n");
        for (i=1; i!=30000; ++i)
        {
                ptr = arry;             /* Set pointer to beginning */
                while (ptr!=arry+128)
                {
                        *ptr = 'X';
                        ++ptr;
                }
        }
        printf("End of Run\n");
}
```

```
/*

                Performance Evaluation Program #5

             Simple Function Calling (no arguments)

                            (BDS C)

*/

main()
{
        int i, j;
        printf("Start of Run\n");
        for (i=1; i!=5000; ++i)
        {
                for (j=1; j!=100; ++j)
                {
                        func1();
                }
        }
        printf("End of Run\n");
}

func1()
{
        func2();
}

func2()
{
}
```

```
/*

                Performance Evaluation Program #6

             Function Calling with Argument Passing

                            (BDS C)

*/

main()
{
        int i, j, k, l;
        printf("Start of Run\n");
        for (i=1; i!=5000; ++i)
        {
                for (j=1; j!=100; ++j)
                {
                        k = func1(j);
                }
        }
        printf("End of Run\n");
}

func1(n)
int n;
{
        int m;
        m = func2(n);
        return m;
}

func2(z)
int z;
{
        return z;
}
```

Small-C Evaluation Programs

```
/*
                 Performance Evaluation Program #1

                       Simple Counting Loops

                             (Small-C)

*/

main()
{
          int i, j;
          puts("Start of Run\n");
          i = 1;
          while (i != 5000)
          {
                    j = 1;
                    while (j != 100)
                    {
                              ++j;
                    }
                    ++i;
          }
          puts("End of Run\n");
}


/*
                 Performance Evaluation Program #2

              Simple Count and Integer Calculation

                             (Small-C)

*/

main()
{
          int i, j, k, l;
          puts("Start of Run");
          i = 1;
          while (i != 30000)
          {
                    j = (5*607+7)/32;
                    k = (j+47)*61;
                    ++i;
          }
          puts("End of Run");
}


/*
                 Performance Evaluation Program #3

                           Conditionals

                             (Small-C)

*/

main()
{
          int i, j;
          puts("Start of Run");
          j = 2500;
          i = 1;
          while (i!=30000)
          {

                    if (i < j)
                    {
                              j = 2500;
                    }
                    else
                    {
                              j = 2500;
                    }

                    if (i > j)
                    {
                              j = 2500;
                    }
                    else
                    {
                              j = 2500;
                    }

                    if (i <= j)
                    {
                              j = 2500;
                    }
                    else
                    {
                              j = 2500;
                    }

                    if (i >= j)
                    {
                              j = 2500;
                    }
                    else
                    {
                              j = 2500;
                    }
                    ++i;

          }
          puts("End of Run");
}
```

```
/*

                Peformance Evaluation Program #4

                        Pointer Operations

                              (Small-C)

*/

main()
{
        char arry[128], *ptr;
        int i;
        puts("Start of Run");
        i = 1;
        while (i != 30000)
        {
                ptr = arry;
                while (ptr!=arry+128)
                {
                        *ptr = 'X';
                        ++ptr;
                }
                ++i;
        }
        puts("End of Run");
}
```

```
/*

                Performance Evaluation Program #5

             Simple Function Calling (no arguments)

                              (Small-C)

*/

main()
{
        int i, j;
        puts ("Start of Run");
        i = 1;
        while (i!=5000)
        {
                j = 1;
                while (j!=100)
                {
                        func1();
                        ++j;
                }
                ++i;
        }
        puts("End of Run");
}
func1()
{
        func2();
}


func2()
{
}
```

```
/*

                Performance Evaluation Program #6

             Function Calling with Argument Passing

                              (Small-C)

*/

main()
{
        int i, j, k, l;
        puts("Start of Run");
        i = 1;
        while (i!=5000)
        {
                j = 1;
                while (j!=100)
                {
                        k = func1(j);
                        ++j;
                }
                ++i;
        }
        puts("End of Run");
}


func1(n)
int n;
{
        int m;
        m = func2(n);
        return m;
}


func2(z)
int z;
{
        return z;
}
```

Tiny-c TWO Evaluation Programs

```
/*              Performance Evaluation Program #1
/*
/*                   Simple Counting Loops
/*
/*                      (tiny-c TWO)

main
[
        int i, j
        pl "Start of Run"
        i = 1;
        while (i != 5000)
        [
                j = 1;
                while (j != 100)
                [
                        ++j;
                ]
                ++i;
        ]
        pl "End of Run"
]
```

```
/*              Performance Evaluation Program #2
/*
/*           Simple Count and Integer Calculation
/*
/*                      (tiny-c TWO)
/*

main
[
        int i, j, k, l
        pl "Start of Run"
        i = 1
        while (i != 30000)
        [
                j = (5*607+7)/32
                k = (j+47)*61
                ++i
        ]
        pl "End of Run"
]
```

```
/*              Performance Evaluation Program #3
/*
/*                       Conditionals
/*
/*                      (tiny-c TWO)
/*

main
[
        int i, j;
        pl "Start of Run"
        j = 2500;
        i = 1;
        while (i!=30000)
        [

                if (i < j)
                [
                        j = 2500;
                ]
                else
                [
                        j = 2500;
                ]

                if (i > j)
                [
                        j = 2500;
                ]
                else
                [
                        j = 2500;
                ]

                if (i <= j)
                [
                        j = 2500;
                ]
                else
                [
                        j = 2500;
                ]

                if (i >= j)
                [
                        j = 2500;
                ]

                else
                [
                        j = 2500;
                ]
                ++i;

         ]
         pl "End of Run"
 ]
```

```
/*              Peformance Evaluation Program #4
/*
/*                       Pointer Operations
/*
/*                         (tiny-c TWO)
/*
main
[
         char arry(128), ptr(0)
         int i
         pl "Start of Run"
         i = 1
         while (i != 300)
         [
                  ptr = arry;
                  while (ptr!=arry+128)
                  [
                           ptr(0) = 'X';    /* Use of pointers in tiny-c /*
                           ++ptr;
                  ]
                  ++i
         ]
         pl "End of Run"
]
```

```
/*              Performance Evaluation Program #5
/*
/*          Simple Function Calling (no arguments)
/*
/*                        (tiny-c TWO)
/*

main
[
         int i, j;
         pl "Start of Run"
         i = 1;
         while (i!=5000)
         [
                  j = 1;
                  while (j!=100)
                  [
                           func1();
                           ++j;
                  ]
                  ++i
         ]
         pl "End of Run"
]


func1
[
         func2();
]


func2
[
]
```

```
/*              Performance Evaluation Program #6
/*
/*          Function Calling with Argument Passing
/*
/*                        (tiny-c TWO)
/*

main
[
         int i, j, k, l;
         pl "Start of Run"
         i = 1;
         while (i!=5000)
         [
                  j = 1;
                  while (j!=100)
                  [
                           k = func1(j);
                           ++j;
                  ]
                  ++i;
         ]
         pl "End of Run"
]


func1
int n
[
         int m;
         m = func2(n);
         return m;
]


func2
int z
[
         return z;
]
```

Whitesmiths Evaluation Programs

```
/*
                Performance Evaluation Program #1

                       Simple Counting Loops

                           (Whitesmiths)

*/

main()
{
         int i, j;
         putfmt("Start of Run\n");
         for (i=1; i!=5000; ++i)
         {
                  for (j=1; j!=100; ++j)
                  {
                  }
         }
         putfmt("End of Run\n");
}
/*

                Performance Evaluation Program #2

            Simple Count and Integer Calculation

                           (Whitesmiths)

*/

main()
{
         int i, j, k, l;
         putfmt("Start of Run\n");
         for (i=1; i!=30000; ++i)
         {
                  j = (5*607+7)/32;
                  k = (j+47)*61;
         }
         putfmt("End of Run\n");
}
 /*

                Performance Evaluation Program #3

                           Conditionals

                           (Whitesmiths)

*/

main()
{
         int i, j;
         putfmt("Start of Run\n");
         j = 2500;
         for (i=1; i!=30000; ++i)
         {

                  if (i < j)
                  {
                           j = 2500;
                  }
                  else
                  {
                           j = 2500;
                  }

                  if (i > j)
                  {
                           j = 2500;
                  }
                  else
                  {
                           j = 2500;
                  }

                  if (i <= j)
                  {
                           j = 2500;
                  }
                  else
                  {
                           j = 2500;
                  }

                  if (i >= j)
                  {
                           j = 2500;
                  }


                   else
                   {
                            j = 2500;
                   }

          }
          putfmt("End of Run\n");
 }

/*

                Peformance Evaluation Program #4

                        Pointer Operations

                           (Whitesmiths)

*/

main()
{
         char arry[128], *ptr;
         int i;
         putfmt("Start of Run\n");
         for (i=1; i!=30000; ++i)
         {

ptr = arry;                 /* Set pointer to beginning */
while (ptr!=arry+128)       /* of array and count till */
{                           /* end.  While is used instead *
         *ptr = 'X';        /* of for as example */
         ++ptr;
```

```
                }
        }
        putfmt("End of Run\n");
}

/*

                Performance Evaluation Program #5

            Simple Function Calling (no arguments)

                        (Whitesmiths)

*/

main()
{
        int i, j;
        putfmt("Start of Run\n");
        for (i=1; i!=5000; ++i)
        {
                for (j=1; j!=100; ++j)
                {
                        func1();
                }
        }
        putfmt("End of Run\n");
}

func1()
{
        func2();
}

func2()
{
}
```

```
/*

                Performance Evaluation Program #6

            Function Calling with Argument Passing

                        (Whitesmiths)

*/

main()
{
        int i, j, k, l;
        putfmt("Start of Run\n");
        for (i=1; i!=5000; ++i)
        {
                for (j=1; j!=100; ++j)
                {
                        k = func1(j);
                }
        }
        putfmt("End of Run\n");
}

func1(n)
int n;
{
        int m;
        m = func2(n);
        return m;
}

func2(z)
int z;
{
        return z;
}
```

A Simple 6 Byte Hexadecimal ASCII Conversion Routine

by Kelly Smith

Only six bytes of 8080 (or Z80) code can perform a hexadecimal (0 to F) to ASCII conversion. Assuming that the hexadecimal digit is in the A register, then:

```
hex$to$ascii:          ; convertlow nibble hex digit
                          in the
                       ; the A req., to ASCII character
                          in the
                       ; A reg.
         adi   90h     ; first add
         daa           ; adjust result, if carry
         aci   40h     ; second add, adjust to ASCII
         daa           ; adjust result, if carry
          .
          .
          .
```

How does it work? There are two main considerations for hexadecimal to ASCII conversion: Is the A register less than ten, or is the A register greater than or equal to ten?

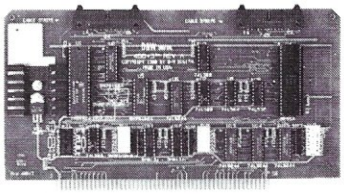
The first DAA instruction (which operates on the lower four bits [low nibble]), adjusts the result of the ADI instruction to less than ten. Then the ACI instruction operates on the upper four bits (high nibble) by adding the carry out of the lower nibble. The second DAA instruction then adjusts the results of the high nibble to less than ten.

If the A register is initially less than ten, the first add results in 9Xh; in this case, the DAA does not affect the A register. The second add (with carry) results in 9Xh+40h =DXh (D Hex = 13 Dec). After the second DAA, the result is 3Xh where "X" is the decimal digits 0 through 9; thus the ASCII representation for decimal digit 9 results in 39 hexadecimal.

If the A register is initially ten or greater, the first add results in 9Xh (same as before), but the result of the foqst DAA is 0Yh (i.e., Yh=Xh-10d) and includes the setting of the carry flag. The next add (with carry) gives us oYh+40h+1=4Zh, where Z=Y+1. The last DAA has no effect. The hexadecimal digits 41 to 46 represent the ASCII alphabetics A to F, for example, let X=10d=Ah, then Y=X-10d=10d-10d=0 and Z=Y+1=0+1=1. The result is 41hexadecimal, the ASCII symbol for "A". This routine is simpler than the explanation! ■

Kelly Smith, 3055 Waco St., Simi Valley, CA 93063.

Reprinted from *CP/M-NET NEWS*



Little-Ada* (Part III)

by Ralph E. Kenyon, Jr.

Little-Ada's Run-time Interpreter and Source Code.

In this installment, the author presents the run-time interpreter and source code for Little-Ada. For background on this subset of the Ada language, please refer to the September/October 1981 issue of *Microsystems*. The November/December 1981 installment of this series discusses the operation of the run-time interpreter and the compiler. Sample programs are also included. The final segment of this article, Part IV, is slated for the March/April 1982 issue, and will present the compiler and object code.

L/I Interpreter Source List

```
Copywrite 1980 by Ralph Kenyon

                   ;Little-Ada L/0 machine interperter
                   ;Edited June 21, 1980
                   ;Copyright 1980 by Ralph Kenyon
                   ;Version 1547 Re-designated L/1 Jan 81
                   ;Stripped down, no debug version

                            REFS SYSTEM.SY  ;Library file
                            REF Warm        ;Warmstart
                            REF WH0         ;Consol Char in
                            REF WH1         ;Consol Char out
                            REF Mss         ;Message writer
                            REF USER        ;Start of user memory
                            REF MEMTOP      ;Last good memory
                            REF Ret         ;Return from overlay
                            REF Dio         ;Disk In/Out
                            REF Err         ;System error handler
                            REF FILE        ;File data buffer
                            REF Ovrto       ;Overlay handler
                            REF CMPTR       ;Command buffer pointer
                            REF Ioret       ;Return from Interupt

                            REFS <#>LOCODE.SY
                   ;Open L/0 code MACRO Library
                            REF LOCODE
                   ;Macro which defines all L/0 code macros.
000D               CR       EQU     13

3200                        ORG USER
3200                        IDNT $,$        ;$ is current value PC

3200  C32F3A                JMP Start
3203  C31135                JMP GO

                            LOCODE
0000                        LIST 0

3206  0D446976     DBZ      DB CR,'Division by zero not defined!',CR,0
320A  6973696F
320E  6E206279
3212  207A6572
3216  6F206E6F
321A  74206465
321E  66696E65
3222  64210D00
```

Ralph E. Kenyon, Jr., 1686 West Main Rd., Portsmouth, RI 02871.

```
3226              Inst    DS 1     ;Instruction register
3227              Base    DS 2     ;Base register
3229              Static  DS'2     ;Static link conversion register
322B              Level   DS 1     ;Level register
322C              AR1     DS 2     ;Arithemetic storage 1
322E              AR2     DS 2     ;Arithemetic storage 2
3230              AR3     DS 2     ;Arithemetic storage 3
3232              TMStack DS 2     ;Stack start
3234              FDB     DS 44    ;File descriptor buffer

3260              IFD     DS 1     ;Input file drive
3261              IFA     DS 2     ;Input file disk address
3263              IFS     DS 2     ;Input file disk sector
3265              IFP     DS 2     ;Input file buffer pointer
3267              IFB     DS 256   ;Input file buffer

3367              OFD     DS 1     ;Output file drive
3368              OFA     DS 2     ;Output file disk address
336A              OFS    'DS 2     ;Output file disk sector
336C              OFP     DS 2     ;Output file buffer pointer
336E              OFB     DS 256   ;Output file buffer
346E              Flag    DS 1     ;Output file in use flag

346F  01          IFflg   DB 1     ;initialize flag
3470  01          OFflg   DB 1     ;initialize flag


3471  0A          Fetch   LDAX B   ;Instruction fetch cycle
3472  03                  INX B
3473  322632              STA Inst
3476  B7                  ORA A
3477  C9                  RET

3478  73          Push    MOV M,E           ;DE to S(t)
3479  2B                  DCX H             ;t+1 to HL
347A  72                  MOV M,D
347B  2B                  DCX H
347C  C9                  RET

347D  23          Pop     INX H             ;S(t) to DE
347E  56                  MOV D,M           ;t-1 to HL
347F  23                  INX H
3480  5E                  MOV E,M
3481  C9                  RET

3482  F5          MinDE   PUSH PSW          ;Two's complement
3483  7A                  MOV A,D           ;of DE. All other
3484  2F                  CMA               ;registers preserved.
3485  57                  MOV D,A
3486  7B                  MOV A,E
3487  2F                  CMA
3488  5F                  MOV E,A
3489  13                  INX D
348A  F1                  POP PSW
348B  C9                  RET

348C  E5          CONV    PUSH H            ;Requires T in DE
348D  CD8234              CALL MinDE        ;(Static)
3490  2A3232              LHLD TMStack
3493  19                  DAD D             ;<[(TMStack)-(Static)]
3494  7C                  MOV A,H           ;We're going to divide by 2
3495  BC                  CMP H             ;(Just reset carry)
```

*Ada is a trademark of the U.S. Department of Defense (Ada Joint Program Office).

NOTE: the DOD does not recognize dialects of the Ada language, whether by supersetting or subsetting.

```
3496  1F                RAR                 ;Puts lo bit in carry
3497  57                MOV D,A             ;Right shifted by 1
3498  7D                MOV A,L             ;Lo byte
3499  1F                RAR                 ;Carry goes into hi bit
349A  5F                MOV E,A             ;(16 bits right shift)
349B  E1                POP H
349C  C9                RET                 ;Result in DE

                  ;This section computes the static link
                  ;by finding the ltack position base for
                  ;L levels down.

349D  F5        GStL    PUSH PSW
349E  E5                PUSH H
349F  3A2632            LDA Inst            ;get & stow level
34A2  E60F      GStL1   ANI 0FH
34A4  2A2732            LHLD Base           ;get & stow base
34A7  222932            SHLD Static
34AA  C3C534            JMP BASE
34AD  2A2932    BASE1   LHLD Static         ;get base
34B0  EB                XCHG
34B1  2A3232            LHLD TMStack
34B4  13                INX D               ;We need to be above by 1
34B5  CD8234            CALL MinDE
34B8  19                DAD D               ;(MEMTOP-2*T)
34B9  19                DAD D               ;stack address now in hl
34BA  CD7D34            CALL Pop            ;Get S(S(t))
34BD  EB                XCHG
34BE  222932            SHLD Static
34C1  3A2B32            LDA Level           ;get level
34C4  3D                DCR A
34C5  322B32    BASE    STA Level
34C8  C2AD34            JNZ BASE1
34CB  EB                XCHG                ;Returns static level in DE
34CC  E1                POP H
34CD  F1                POP PSW
34CE  C9                RET

34CF  1E02      Out2    MVI E,2             ;Output file already exists
34D1  C3D634            JMP Out0
34D4  1E03      Out3    MVI E,3             ;Input file not specified
34D6  1607      Out0    MVI D,7
34D8  C30F04    Out     JMP Err

34DB  3EE0      Gf      MVI A,0E0H
34DD  CD1204    Gf1     CALL Ovrto
34E0  47666964          DB 'Gfid'
34E4  C9                RET

                  ;Parameters for Dio set up by start code
                  ;Here's where we get the file to be
                  ;interpretered

34E5  CD0604    GETP    CALL Dio            ;Go get it.
34E8  DAD834            JC Out              ;Something Wrong!
34EB  212F3A            LXI H,Pgmaddr       ;get the program
34EE  E5                PUSH H
34EF  C1                POP B               ;Set TMPC to first byte
34F0  2A3232            LHLD TMStack        ;Set initialize TMSP
34F3  110000            LXI D,0             ;First position on stack for
34F6  CD7834            CALL Push           ;Character in/out
34F9  CD7834            CALL Push           ;Static link
34FC  13                INX D
34FD  EB                XCHG
34FE  222732            SHLD Base           ;set Base 1st
3501  EB                XCHG
3502  CD7834            CALL Push           ;Dynamic link same
3505  112E3A            LXI D,Origin        ;addr of that 'hlt' byte
3508  CD7834            CALL Push
350B  CDFF37            CALL INB
350E  CD5039            CALL OUTB

                  ;This routine sets itself up as a return address

3511  E5        GO      PUSH H              ;Return to here
3512  211135            LXI H,GO
3515  E3                XTHL                ;Put our addr on stack
3516  CD7134            CALL Fetch
3519  17                RAL
351A  D2A635            JNC branch          ;0 means br or bnz
351D  17                RAL
351E  D26935            JNC oprlic
3521  17                RAL
3522  D8                RC                  ;111XXXXX is NOP
3523  CD9D34            CALL GStL           ;For both lad & call
3526  17                RAL                 ;Now which one
3527  DA3C35            JC Call             ;do we have?

                  ;Here we have to get the address from
                  ;the program immediate data (two bytes)

352A  E5        Lad     PUSH H
352B  2A2932            LHLD Static
352E  CD7134            CALL Fetch
3531  57                MOV D,A             ;Address hi byte
3532  CD7134            CALL Fetch
3535  5F                MOV E,A             ;Address lo byte
3536  19                DAD D               ;Add in the stack base
3537  EB                XCHG                ;put it in DE
3538  E1                POP H
3539  C37834            JMP Push            ;Let push return

                  ;This routine puts links on stack
                  ;followed by return address

353C  E5        Call    PUSH H              ;We need TMSP later
353D  EB                XCHG
353E  2A2932            LHLD Static
3541  EB                XCHG
3542  CD7834            CALL Push           ;Static link first
3545  EB                XCHG
3546  2A2732            LHLD Base
3549  EB                XCHG
```

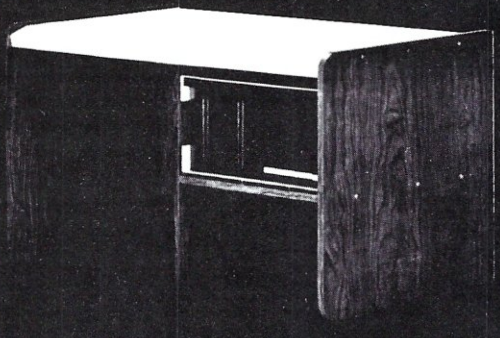
```
354A  CD7834            CALL Push           ;Dynamic link second
354D  E3                XTHL                ;TMSP to stack
354E  EB                XCHG
354F  CD8C34            CALL CONV
3552  EB                XCHG
3553  222732            SHLD Base           ;Set new base
3556  CD7134            CALL Fetch          ;lets get that address
3559  57                MOV D,A
355A  CD7134            CALL Fetch
355D  5F                MOV E,A
355E  212F3A            LXI H,Pgmaddr
3561  19                DAD D
3562  E3                XTHL                ;Addr to top of stack
3563  C5                PUSH B
3564  D1                POP D
3565  C1                POP B
3566  C37834            JMP Push            ;return address

3569  17        oprlic  RAL                 ;Check next bit for oprlic
356A  DA8135            JC Lic

                  ;For opr, we must get last 5 bits from inst
                  ;We'll use a computed goto to get the
                  ;routine for the sub-operation.

356D  3A2632    opr     LDA Inst
3570  E61F              ANI 1FH
3572  87                ADD A               ;Times 2
3573  5F                MOV E,A
3574  1600              MVI D,0
3576  E5                PUSH H              ;save TMSP
3577  21CB35            LXI H,Jtbl          ;Jmp table
357A  19                DAD D               ;add position
357B  5E                MOV E,M
357C  23                INX H
357D  56                MOV D,M
357E  EB                XCHG                ;addr to HL
357F  E3                XTHL                ;addr to stack
3580  C9                RET                 ;Jump to addr

                  ;Now we've got to sort out the number of
                  ;bytes used for the constant in this lic

3581  17        Lic     RAL
3582  DA8F35            JC Lic1
3585  3A2632            LDA Inst            ;1 byte
3588  E60F              ANI 0FH
358A  1600              MVI D,0
358C  C3A235            JMP lic4
358F  17        Lic1    RAL
3590  DA9B35            JC lic2
3593  3A2632            LDA Inst            ;2 byte
3596  E607              ANI 7
3598  C39E35            JMP lic3
359B  CD7134    lic2    CALL Fetch          ;3 byte
359E  57        lic3    MOV D,A
359F  CD7134            CALL Fetch
35A2  5F        lic4    MOV E,A
35A3  C37834            JMP Push            ;let push RET for us

35A6  17        branch  RAL
35A7  D2B935            JNC Br
35AA  CD7D34            CALL Pop
35AD  7A                MOV A,D
35AE  B7                ORA A
35AF  C2B935            JNZ Br              ;(bnz)
35B2  83                ADD E
35B3  C2B935            JNZ Br              ;(bnz)
35B6  C37134            JMP Fetch           ;Skip this byte
                                            ;let Fetch return

35B9  3A2632    Br      LDA Inst
35BC  E63F              ANI 3FH             ;Kill opcode
35BE  57                MOV D,A             ;Hi addr
35BF  CD7134            CALL Fetch          ;rest of addr
35C2  5F                MOV E,A             ;Lo addr
35C3  E5                PUSH H
35C4  212F3A            LXI H,Pgmaddr       ;Adjust for program
35C7  19                DAD D               ;load address
35C8  E3                XTHL
35C9  C1                POP B
35CA  C9                RET

35CB  0B36      Jtbl    DW Halt             ;0

                  ; Halt closes both the input and the
                  ; output files before invoking Exec.
                  ; The input and output file setup routines
                  ; are restored to IFR and OFR also.

35CD  1636              DW addsub           ;1
35CF  1636              DW addsub           ;2
35D1  2D36              DW muldiv           ;3
35D3  2D36              DW muldiv           ;4
35D5  F236              DW Mod              ;5
35D7  3637              DW Neg              ;6
35D9  3F37              DW Not              ;7
35DB  8837              DW Sete             ;8
35DD  A837              DW Setlg            ;9
35DF  A837              DW Setlg            ;A
35E1  5737              DW Swap             ;B
35E3  6837              DW retn             ;C
35E5  CB37              DW Rav              ;D
35E7  DF37              DW Sto              ;E
35E9  F637              DW inc              ;F
35EB  FF37      IFR     DW INB              ;10

                  ; INB sets up the input file data for Dio
                  ; and puts the address of Inb into IFR.
                  ; If a file is not selected, INB puts the
                  ; address of Cinb into IFR (input from consol)

35ED  5039      OFR     DW OUTB             ;11
```

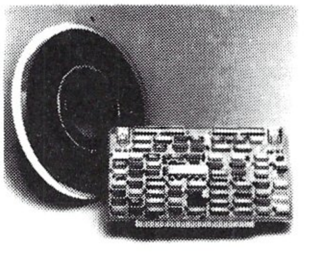
```
                 ; OUTB sets up the output file data for Dio
                 ; and puts the address of Outb into OFR.
                 ; If a file is not selected, OUTB puts the
                 ; address of Coutb into OFR (output to consol)

                 ;These remaining are all treated as nop

35EF  2805               DW Ret         ;12 insurance
35F1  2805               DW Ret         ;13
35F3  2805               DW Ret         ;14
35F5  2805               DW Ret         ;15
35F7  2805               DW Ret         ;16
35F9  2805               DW Ret         ;17
35FB  2805               DW Ret         ;18
35FD  2805               DW Ret         ;19
35FF  2805               DW Ret         ;1A
3601  2805               DW Ret         ;1B
3603  2805               DW Ret         ;1C
3605  2805               DW Ret         ;1D
3607  2805               DW Ret         ;1E
3609  2805               DW Ret         ;1F

360B  CDE539     Halt    CALL TURNOFF   ;Close open output file
360E  21FF37             LXI H,INB      ;Restore Input file
3611  22EB35             SHLD IFR       ;Open sequence
3614  D1                 POP D          ;Clean up stack
3615  C9                 RET

3616  CD7D34     addsub  CALL Pop       ;S(t)
3619  D5                 PUSH D
361A  CD7D34             CALL Pop       ;S(t-1)
361D  E3                 XTHL           ;S(t) to HL
361E  EB                 XCHG           ;S(t) to DE
361F  3A2632             LDA Inst
3622  E602               ANI 2          ;is it a subtract?
3624  C48234             CNZ MinDE
3627  19                 DAD D          ;S(t-1)-S(t) IN HL
3628  EB                 XCHG
3629  E1                 POP H          ;Get TMSP back
362A  C37834             JMP Push       ;let push return for us

362D  CD7D34     muldiv  CALL Pop
3630  EB                 XCHG
3631  222C32             SHLD AR1
3634  EB                 XCHG
3635  CD7D34             CALL Pop
3638  EB                 XCHG
3639  222E32             SHLD AR2
363C  3A2632             LDA Inst
363F  E604               ANI 4          ;not multiply?
3641  CC4E36             CZ MULT
3644  C49936             CNZ DIVD
3647  2A3032             LHLD AR3
364A  EB                 XCHG
364B  C37834             JMP Push       ;let push return for us

364E  F5         MULT    PUSH PSW       ;16 bit multiply
364F  C5                 PUSH B         ;with no overflow test
3650  D5                 PUSH D         ;returns product mod 10000H
3651  E5                 PUSH H
3652  2A2C32             LHLD AR1
3655  7C                 MOV A,H
3656  B7                 ORA A
3657  C25F36             JNZ MULT1
365A  85                 ADD L
365B  CA9036             JZ MULT7
365E  EB                 XCHG
365F  2A2E32     MULT1   LHLD AR2
3662  7C                 MOV A,H
3663  B7                 ORA A
3664  C26B36             JNZ MULT2
3667  85                 ADD L
3668  CA9036             JZ MULT7
366B  4C         MULT2   MOV C,H        ;save hi byte
366C  7D                 MOV A,L        ;do lo byte
366D  210000             LXI H,0
3670  0608               MVI B,8
3672  0F         MULT3   RRC
3673  D27736             JNC MULT4
3676  19                 DAD D
3677  EB         MULT4   XCHG
3678  29                 DAD H
3679  EB                 XCHG
367A  05                 DCR B
367B  C27236             JNZ MULT3
367E  79                 MOV A,C        ;now do hi byte
367F  0608               MVI B,8
3681  0F         MULT5   RRC
3682  D28636             JNC MULT6
3685  19                 DAD D
3686  EB         MULT6   XCHG
3687  29                 DAD H
3688  EB                 XCHG
3689  05                 DCR B
368A  C28136             JNZ MULT5
368D  C39336             JMP MULT8
3690  210000     MULT7   LXI H,0
3693  223032     MULT8   SHLD AR3
3696  C36400             JMP Ioret

3699  F5         DIVD    PUSH PSW
369A  C5                 PUSH B
369B  D5                 PUSH D
369C  E5                 PUSH H
369D  010000             LXI B,0        ;Result goes here
36A0  2A2C32             LHLD AR1
36A3  7C                 MOV A,H        ;lets see if
36A4  B7                 ORA A          ;the idiot wants
36A5  C2AC36             JNZ DIVD1      ;to divide by
36A8  85                 ADD L          ;zero.
36A9  CAE536             JZ DBZER       ;He does!

36AC  EB         DIVD1   XCHG           ;nope, so get
36AD  2A2E32             LHLD AR2       ;dividend
```

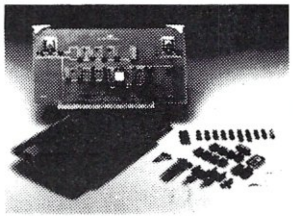
```
36B0  7A                 MOV A,D        ;If it's
36B1  B7                 ORA A          ;zero
36B2  C2BF36             JNZ DIVD2      ;then
36B5  85                 ADD E          ;result's
36B6  C2BF36             JNZ DIVD2      ;also
36B9  210000     DIVD7   LXI H,0        ;zero
36BC  C3DF36             JMP DIVD6

36BF  7C         DIVD2   MOV A,H
36C0  BA                 CMP D
36C1  DADD36             JC DIVD4
36C4  CACB36             JZ DIVD3
36C7  03                 INX B
36C8  C3D436             JMP SUBT
36CB  7D         DIVD3   MOV A,L
36CC  BB                 CMP E
36CD  DADD36             JC DIVD4
36D0  03                 INX B
36D1  CADD36             JZ DIVD4
36D4  D5         SUBT    PUSH D
36D5  CD8234             CALL MinDE
36D8  19                 DAD D
36D9  D1                 POP D
36DA  C3BF36             JMP DIVD2
36DD  C5         DIVD4   PUSH B
36DE  E1                 POP H
36DF  223032     DIVD6   SHLD AR3
36E2  C36400             JMP Ioret

36E5  CDEB36     DBZER   CALL DBZ1
36E8  C3B936             JMP DIVD7

36EB  210632     DBZ1    LXI H,DBZ
36EE  CD0C04             CALL Msg
36F1  C9                 RET

36F2  CD7D34     Mod     CALL Pop       ;S(t) to DE
36F5  D5                 PUSH D         ;S(t) to top of stack
36F6  CD7D34             CALL Pop       ;S(t-1) to DE
36F9  E3                 XTHL           ;S(t) to HL
36FA  7C                 MOV A,H        ;lets see if
36FB  B7                 ORA A          ;the idiot wants
36FC  C20937             JNZ Mod1       ;to divide by
36FF  85                 ADD L          ;zero.
3700  C20937             JNZ Mod1
3703  CDEB36             CALL DBZ1
3706  C32D37             JMP Mod3       ;He does!

3709  7A         Mod1    MOV A,D        ;see if we
370A  B7                 ORA A          ;start with
370B  C21D37             JNZ TEST       ;zero
370E  83                 ADD E
370F  C21D37             JNZ TEST
3712  C32D37             JMP Mod3

3715  EB         SUBTR   XCHG
3716  D5                 PUSH D         ;Save
3717  CD8234            ·CALL MinDE
371A  19                 DAD D          ;Add -DE
371B  D1                 POP D          ;Restore
371C  EB                 XCHG
371D  7A         TEST    MOV A,D        ;Hi byte of S(t)
371E  BC                 CMP H
371F  DA3037             JC Done        ;Hi byte of S(t-1)
                                        ;<Hi byte of S(t)
3722  C21537             JNZ SUBTR      ;its bigger
3725  7B                 MOV A,E        ;It's equal so
3726  BD                 CMP L          ;Check lo byte
3727  DA3037             JC Done
372A  C21537             JNZ SUBTR      ;its bigger
372D  110000     Mod3    LXI D,0        ;its equal
3730  EB         Done    XCHG
3731  E3                 XTHL
3732  D1                 POP D
3733  C37834             JMP Push       ;let push return for us

3736  CD7D34     Neg     CALL Pop       ;S(t) to DE
3739  CD8234             CALL MinDE
373C  C37834             JMP Push       ;DE to S(t) let push ret

373F  CD7D34     Not     CALL Pop       ;look
3742  7A                 MOV A,D        ;hi byte
3743  B7                 ORA A          ;set flags
3744  C25137             JNZ Not2
3747  83         Not1    ADD E          ;lo byte
3748  C25137             JNZ Not2
374B  110100             LXI D,1        ;its Zero so change result
374E  C37834             JMP Push
3751  110000     Not2    LXI D,0
3754  C37834             JMP Push       ;onto stack let
                                        ;push ret for us
3757  CD7D34     Swap    CALL Pop       ;S(t)
375A  D5                 PUSH D         ;to TOS
375B  CD7D34             CALL Pop       ;S(t-1) to DE
375E  E3                 XTHL           ;S(t) TO HL, t-1 to TOS
375F  EB                 XCHG           ;S(t) to DE, S(t-1) to HL
3760  E3                 XTHL           ;t-1 to HL, S(t-1) to TOS
3761  CD7834             CALL Push      ;S(t-1) to TOS
3764  D1                 POP D          ;S(t-1) to DE
3765  C37834             JMP Push       ;S(t-1) to TMS
                                        ;let push return for us.

3768  2A2732     retn    LHLD Base
376B  110300             LXI D,3
376E  19                 DAD D
376F  29                 DAD H
3770  EB                 XCHG
3771  CD8234             CALL MinDE
3774  2A3232             LHLD TMStack
3777  19                 DAD D
3778  CD7D34             CALL Pop       ;TMPC
377B  D5                 PUSH D
377C  C1                 POP B
377D  CD7D34             CALL Pop       ;Dynamic link
```

```
3780  EB                  XCHG
3781  222732              SHLD Base
3784  EB                  XCHG
3785  23                  INX H               ;We don't need that
3786  23                  INX H               ;static link now
3787  C9                  RET

3788  CD7D34    Sete      CALL Pop
378B  D5                  PUSH D
378C  CD7D34              CALL Pop
378F  E3                  XTHL
3790  7A                  MOV A,D
3791  BC                  CMP H
3792  C2A137              JNZ SETE1
3795  7B                  MOV A,E
3796  BD                  CMP L
3797  C2A137              JNZ SETE1
379A  110100              LXI D,1             ;they're equal
379D  E1                  POP H
379E  C37834              JMP Push            ;let push return for us

37A1  110000    SETE1     LXI D,0
37A4  E1                  POP H
37A5  C37834              JMP Push            ;let push return for us

37A8  CD7D34    Setls     CALL Pop
37AB  D5                  PUSH D              ;S(t) to TOS
37AC  CD7D34              CALL Pop            ;S(t-1) to DE
37AF  E3                  XTHL                ;S(t) to HL
37B0  3A2632              LDA Inst
37B3  E602                ANI 2               ;Setst?
37B5  CAB937              JZ Set1
37B8  EB                  XCHG                ;Reverse for Setst
37B9  CD8234    Set1      CALL MinDE          ;-S(t-1)
37BC  19                  DAD D               ;Want 0<S(t)-S(t-1)
37BD  2B                  DCX H               ;Sign test uses >= 0
37BE  7C                  MOV A,H             ;Look at sign
37BF  B7                  ORA A               ;Set flags
37C0  E1                  POP H               ;TMSP
37C1  110100              LXI D,1             ;Assume true
37C4  F2C837              JP Set2             ;Jump if true
37C7  1B                  DCX D               ;Falls thru if false
37C8  C37834    Set2      JMP Push            ;Let Push return for us

                ;Note: RAV assumes that the address on the stack
                ;is a relative address from the TM stack pointer
                ;with 1 for each 16 bit push or pop.  We multiply
                ;the two's complement by 2 and add it to
                ;the address in TMStack (Top of memory)

37CB  CD7D34    Rav       CALL Pop            ;Get S(t)
37CE  E5                  PUSH H              ;Save SP
37CF  2A3232              LHLD TMStack
37D2  13                  INX D               ;We need to be above by 1
37D3  CD8234              CALL MinDE
37D6  19                  DAD D               ;(MEMTOP-2*T)
37D7  19                  DAD D               ;stack address now in hl
37D8  CD7D34              CALL Pop            ;Get S(S(t))
37DB  E1                  POP H               ;Restore TMSP
37DC  C37834              JMP Push            ;S(t):=S(S(t))

37DF  CD7D34    Sto       CALL Pop            ;S(t) to be stowed
37E2  D5                  PUSH D              ;save it
37E3  CD7D34              CALL Pop            ;address to stow S(t) in
37E6  E3                  XTHL                ;(We'll want S(t) first)
37E7  E5                  PUSH H              ;Need to use HL
37E8  CD8234              CALL MinDE          ;Convert Stack
37EB  2A3232              LHLD TMStack        ;address
37EE  19                  DAD D               ;(MEMTOP-2*T)
37EF  19                  DAD D               ;stack address now in hl
37F0  D1                  POP D               ;Get S(t)
37F1  CD7834              CALL Push           ;S(S(T-1)):=S(T)
37F4  E1                  POP H               ;T-2 to TMSP
37F5  C9                  RET

37F6  CD7D34    Inc       CALL Pop            ;S(t) to de, t-1 in HL
37F9  CD8234              CALL MinDE
37FC  19                  DAD D
37FD  19                  DAD D               ;S(t)+t-1 to HL
37FE  C9                  RET

37FF  E5        INB       PUSH H              ;Save VMSP
3800  C5                  PUSH B              ;Save VMPC
3801  216F38              LXI H,Ifpr          ;get one from him.

3804  11CB2D    IFR1      LXI D,FILE          ;File descriptor buffer
3807  014441              LXI B,'AD'          ;Default file extension
380A  CDDB34              CALL Gf
380D  D28C38              JNC IFR2            ;Gfid found the file
                                              ;so go read it

3810  AF                  XRA A               ;Checks for error
3811  82                  ADD D               ;code 0503H
3812  FE05                CPI 5
3814  C20F04              JNZ Err             ;Wrong one
3817  83                  ADD E
3818  FE08                CPI 8               ;adds up to 8
381A  C20F04              JNZ Err             ;No good!
381D  212638              LXI H,Cinb          ;Set up to get input
3820  22EB35              SHLD IFR            ;from the consol
3823  C1                  POP B               ;VMPC
3824  E1                  POP H               ;VMSP
3825  C9                  RET

                ; Additional inputs Jump to here

3826  CD200C    Cinb      CALL WHO            ;We're inputting from
3829  E5                  PUSH H              ;the consol
382A  2A3232              LHLD TMStack        ;Where it goes
382D  77                  MOV M,A             ;Put it in
382E  E1                  POP H               ;VMSP
382F  C9                  RET

3830  0D546865  Ifprn     DB CR,'The input file''s empty.'
```

```
3834  20696E70
3838  75742066
383C  696C6527
3840  7320656D
3844  7074792E
3848  0D576861   DB CR,'What''s the continuation file''s name? ',0
384C  74277320
3850  74686520
3854  636F6E74
3858  696E7561
385C  74696F6E
3860  2066696C
3864  65277320
3868  6E616D65
386C  3F2000
386F  57686174  Ifpr      DB 'What''s the input file name? ',0
3873  27732074
3877  68652069
387B  6E707574
387F  2066696C
3883  65206E61
3887  6D653F20
388B  00

388C  21CB2D    IFR2      LXI H,FILE          ;READ starts here


388F  7E                  MOV A,M
3890  E607                ANI 7               ;trim down to drive no.
3892  326032              STA IFD             ;Drive number
3895  23                  INX H
3896  7E                  MOV A,M             ;FDE flag byte
3897  E61F                ANI 1FH             ;trim to file size
3899  C603                ADI 3               ;point past extension
389B  5F                  MOV E,A             ;Put into DE
389C  1600                MVI D,0
389E  19                  DAD D               ;Add to Address in HL
389F  EB                  XCHG                ;FDA pointer now in DE
38A0  216132              LXI H,IFA           ;Where the addresses go
38A3  0E04                MVI C,4             ;4 bytes to copy
38A5  1A        CIFD      LDAX D              ;Get the data
38A6  77                  MOV M,A             ;from the FDB (FILE)
38A7  23                  INX H               ;and copy into the
38A8  13                  INX D               ;areas for our Dio
38A9  0D                  DCR C               ;routines
38AA  C2A538              JNZ CIFD            ;More to copy
38AD  216733              LXI H,IFB+100H      ;Reset the
38B0  226532              SHLD IFP            ;buffer pointer too
38B3  21BC38              LXI H,Inb           ;Furthur calls to Reader
38B6  22EB35              SHLD IFR            ;the reader
38B9  C1                  POP B               ;VMPC
38BA  E1                  POP H               ;VMSP
38BB  C9                  RET

                ; Routine to input from an open file

38BC  E5        Inb       PUSH H              ;Save VMSP
38BD  C5                  PUSH B              ;Save VMPC
38BE  2A6532    RD1       LHLD IFP
38C1  116733              LXI D,IFB+100H
38C4  7C                  MOV A,H
38C5  BA                  CMP D
38C6  C2CE38              JNZ RD2
38C9  7D                  MOV A,L
38CA  BB                  CMP E
38CB  CADA38              JZ RD3
38CE  7E        RD2       MOV A,M
38CF  23                  INX H
38D0  226532              SHLD IFP
38D3  C1                  POP B               ;VMPC
38D4  2A3232              LHLD TMStack        ;Here's where
38D7  77                  MOV M,A             ;we put it
38D8  E1                  POP H               ;VMSP
38D9  C9                  RET

38DA  2A6332    RD3       LHLD IFS
38DD  7C                  MOV A,H
38DE  B7                  ORA A
38DF  C2EC38              JNZ RD4
38E2  B5                  ORA L
38E3  C2EC38              JNZ RD4

38E6  213038              LXI H,Ifprn
38E9  C30438              JMP IFR1

38EC  2B        RD4       DCX H               ;Got to get another
38ED  226332              SHLD IFS            ;sector from disk
38F0  216732              LXI H,IFB
38F3  226532              SHLD IFP
38F6  D5                  PUSH D
38F7  EB                  XCHG
38F8  2A6132              LHLD IFA            ;Get disk address
38FB  23                  INX H               ;update for next time
38FC  226132              SHLD IFA            ;and save
38FF  2B                  DCX H               ;back to the one we want
3900  C5                  PUSH B              ;going to preserve B
3901  0601                MVI B,1             ;Read
3903  3A6032              LDA IFD             ;Drive for input file
3906  4F                  MOV C,A             ;into C
3907  3E01                MVI A,1             ;1 sector
3909  CD0604              CALL Dio            ;Get it
390C  C1                  POP B               ;restore
390D  D1                  POP D               ;this too
390E  D2BE38              JNC RD1             ;Now we can get another byte
3911  C30F04              JMP Err

3914  57686174  Ofpr      DB 'What''s the output file name? ',0
3918  27732074
391C  6865206F
3920  75747075
3924  74206669
3928  6C65206E
392C  616D653F
3930  2000
```







```
3932  FE03    CK1     CPI 3          ;Now lets check
3934  C20F04          JNZ Err        ;for the 0503 error
3937  82              ADD D
3938  FE08            CPI 8          ;adds up to 8
393A  C20F04          JNZ Err        ;No good!
393D  214639          LXI H,Coutb
3940  22ED35          SHLD OFR
3943  C1              POP B          ;VMPC
3944  E1              POP H          ;VMSP
3945  C9              RET

              ; Ouputs jump to here

3946  E5      Coutb   PUSH H         ;We're outputting to the consol
3947  2A3232          LHLD TMStack
394A  7E              MOV A,M
394B  CD240C          CALL WH1
394E  E1              POP H
394F  C9              RET

3950  E5      OUTB    PUSH H         ;Save VMSP
3951  C5              PUSH B         ;Save VMPC
3952  211439          LXI H,Ofpr     ;get one from him.
3955  113432          LXI D,FDB      ;File descriptor buffer
3958  014941          LXI B,'AI'     ;('AI' is default ext)
395B  CDDB34          CALL Gf
395E  D2CF34          JNC Out2
3961  AF              XRA A          ;Checks for error
3962  83              ADD E          ;code 0300H or 0503H
3963  C23239          JNZ CK1        ;Does not return
3966  82              ADD D          ;unless one was
3967  FE03            CPI 3          ;found. Sets CARRY
3969  C20F04          JNZ Err        ;Need to have
                                     ;a 0300 error
396C  213432          LXI H,FDB      ;We need to save this
                                     ;for close
396F  7E              MOV A,M
3970  E607            ANI 7          ;trim down to drive no.
3972  326733          STA OFD        ;Drive number
3975  23              INX H
3976  7E              MOV A,M        ;FDE flag byte
3977  E61F            ANI 1FH        ;trim to file size
3979  C603            ADI 3          ;point past extension
397B  5F              MOV E,A        ;Put into DE
397C  1600            MVI D,0
397E  19              DAD D          ;Add to Address in HL
397F  EB              XCHG           ;FDA pointer now in DE
3980  216833          LXI H,OFA      ;Where the addresses go
3983  0E04            MVI C,4        ;4 bytes to copy
3985  1A      COFD    LDAX D         ;Get the data
3986  77              MOV M,A        ;from the FDB
3987  23              INX H          ;and copy into the
3988  13              INX D          ;areas for our Dio
3989  0D              DCR C          ;routines
398A  C28539          JNZ COFD       ;More to copy
398D  216E33          LXI H,OFB      ;Reset the
3990  226C33          SHLD OFP       ;buffer pointer too
3993  219C39          LXI H,Outb     ;characters thru
3996  22ED35          SHLD OFR
3999  C1              POP B          ;VMPC
399A  E1              POP H          ;VMSP
399B  C9              RET

              ; Routine to output to an open file
              ; thru calls to Outb

399C  F5      Outb    PUSH PSW       ;For writing
399D  C5              PUSH B
399E  D5              PUSH D
399F  E5              PUSH H
39A0  216400          LXI H,Ioret

39A3  E5              PUSH H
39A4  2A3232          LHLD TMStack   ;Get the char
39A7  7E              MOV A,M

              ;The rest of this is called as a subroutine for
              ;filling up the last sector with zeros also.

39A8  2A6C33  Store   LHLD OFP
39AB  77              MOV M,A        ;put char in buffer
39AC  116E34          LXI D,Flag
39AF  1A              LDAX D
39B0  B7              ORA A
39B1  C2B639          JNZ Store1
39B4  3D              DCR A          ;We've been had!
39B5  12              STAX D
39B6  23      Store1  INX H          ;bump pointer
39B7  226C33          SHLD OFP
39BA  116E33          LXI D,OFB
39BD  25              DCR H
39BE  7C              MOV A,H
39BF  BA              CMP D
39C0  C0              RNZ
39C1  7D              MOV A,L
39C2  BB              CMP E
39C3  C0              RNZ

              ;pointer now points at OFB so do DIO.

39C4  226C33          SHLD OFP       ;DE points at OFB

39C7  2A6A33          LHLD OFS       ;Number of sectors
39CA  23              INX H          ;One more
39CB  226A33          SHLD OFS
39CE  2A6833          LHLD OFA       ;Disk address
39D1  23              INX H          ;Up date for next time
39D2  226833          SHLD OFA
39D5  2B              DCX H          ;Here's where we write
```

```
39D6  3A6733           LDA OFD          ;Drive
39D9  4F               MOV C,A          ;Drive no.
39DA  0600             MVI B,0          ;Write
39DC  3E01             MVI A,1          ;one sector
39DE  CD0604           CALL Dio
39E1  DA0F04           JC Err
39E4  C9               RET

                ; Routines for closing the file


39E5  E5        TURNOFF PUSH H          ;Save VMSP
39E6  C5               PUSH B           ;Save VMPC
39E7  3A6E34           LDA Flag         ;See if we're
                                        ;still Virgin.
39EA  B7               ORA A            ;(Also for closing
39EB  CA213A           JZ T01           ;a read file.)
39EE  3A6C33    Fill   LDA OFP          ;Not virgin,

39F1  FE6E             CPI OFB AND 0FFH
39F3  3E00             MVI A,0
39F5  CAFE39           JZ Close1
39F8  CDA839           CALL Store       ;fill up last sector
39FB  C3EE39           JMP Fill         ;with zeros

39FE  213532    Close1 LXI H,FDB+1
3A01  7E               MOV A,M
3A02  E61F             ANI 1FH          ;strip down to length
3A04  C605             ADI 5            ;Point past ext and FDA
3A06  5F               MOV E,A
3A07  1600             MVI D,0
3A09  19               DAD D
3A0A  EB               XCHG             ;adr of DNS now in DE
3A0B  2A6A33           LHLD OFS
3A0E  EB               XCHG
3A0F  73               MOV M,E
3A10  23               INX H
3A11  72               MOV M,D          ;length now updated
3A12  213432           LXI H,FDB
3A15  7E               MOV A,M
3A16  E67F             ANI 7FH
3A18  77               MOV M,A
3A19  3E01             MVI A,1          ;enter new output
                                        ;file in directory
3A1B  CDDD34           CALL Gf1
3A1E  DA0F04           JC Err
3A21  AF        T01    XRA A            ;Virgin exit.
3A22  326E34           STA Flag
3A25  215039    Out1   LXI H,OUTB       ;Restore calling address
3A28  22ED35           SHLD OFR         ;to open a file
3A2B  C1               POP B            ;VMPC
3A2C  E1               POP H            ;VMSP
3A2D  C9               RET

                Origin  hlt     ;LO MACRO instruction
3A2E  80        Origin  DB 80H
3A2F            Pgmaddr EQU $

                ; We load the executable file on top
                ;of the Start code !!


3A2F  2A802D    Start  LHLD MEMTOP
3A32  223232           SHLD TMStack
3A35  210032           LXI H,USER
3A38  36C9             MVI M,RET        ;Don't START again
3A3A  2AC72D           LHLD CMPTR       ;Cmd pointer
3A3D  7E               MOV A,M
3A3E  FE0D             CPI CR
3A40  CAD434           JZ Out3
3A43  113432           LXI D,FDB        ;File descriptor block
                                        ;built by Gfid
3A46  01304C           LXI B,4C30H      ;L/0 extension for
                                        ;default is L0

                ; We've reached the end of the input file
                ; so, we ask for another one

3A49  3E60             MVI A,60H
3A4B  CDDD34           CALL Gf1
3A4E  DAD834           JC Out           ;Something Wrong!
3A51  213432           LXI H,FDB
3A54  7E               MOV A,M
3A55  E607             ANI 7            ;Kill flags
3A57  77               MOV M,A
3A58  23               INX H            ;Move up to FDE flags.
3A59  7E               MOV A,M
3A5A  E61F             ANI 1FH          ;Kill flags
3A5C  C603             ADI 3            ;Point past ext
3A5E  5F               MOV E,A
3A5F  1600             MVI D,0
3A61  19               DAD D            ;Addr of FDA
3A62  5E               MOV E,M
3A63  23               INX H
3A64  56               MOV D,M
3A65  23               INX H
3A66  3A3432           LDA FDB
3A69  4F               MOV C,A          ;Drive to C
3A6A  0601             MVI B,1          ;Read
3A6C  7E               MOV A,M          ;DNS
3A6D  EB               XCHG             ;FDA to HL
3A6E  112F3A           LXI D,Pgmaddr    ;Where to put it
3A71  C3E534           JMP GETP

                       END


Error total =      0
```

```
             Macros defined in this assembly:

LOCODE          add             bnz             br
call            div             hlt             inb
inc             lad             lic             mod
mul             neg             nop             not
outb            rav             ret             sete
setgt           setlt           sto             sub
swap

             Labels defined in this assembly:

AR1      322C AR2      322E AR3      3230 BASE     34C5
BASE1    34AD Base     3227 Br       35B9 CIFD     38A5
CK1      3932 CMPTR    2DC7 COFD     3985 CONV     348C
CR       000D Call     353C Cinb     3826 Close1   39FE
Coutb    3946 DBZ      3206 DBZ1     36EB DBZER    36E5
DIVD     3699 DIVD1    36AC DIVD2    36BF DIVD3    36CB
DIVD4    36DD DIVD6    36DF DIVD7    36B9 Dio      0406
Done     3730 Err      040F FDB      3234 FILE     2DCB
Fetch    3471 Fill     39EE Flag     346E GETP     34E5
GO       3511 GStL     349D GStL1    34A2 Gf       34DB
Gf1      34DD Halt     360B IFA      3261 IFB      3267
IFD      3260 IFP      3265 IFR      35EB IFR1     3804
IFR2     388C IFS      3263 IFflg    346F INB      37FF
Ifpr     386F Ifprn    3830 Inb      38BC Inc      37F6
Inst     3226 Ioret    0064 Jtbl     35CB Lad      352A
Level    322B Lic      3581 Lic1     358F MEMTOP   2D80
MULT     364E MULT1    365F MULT2    366B MULT3    3672
MULT4    3677 MULT5    3681 MULT6    3686 MULT7    3690
MULT8    3693 MinDE    3482 Mod      36F2 Mod1     3709
Mod3     372D Msg      040C Neg      3736 Not      373F
Not1     3747 Not2     3751 OFA      3368 OFB      336E
OFD      3367 OFP      336C OFR      35ED OFS      336A
OFflg    3470 OUTB     3950 Ofpr     3914 Origin   3A2E
Out      34D8 Out0     34D6 Out1     3A25 Out2     34CF
Out3     34D4 Outb     399C Ovrto    0412 Pgmaddr  3A2F
Pop      347D Push     3478 RD1      38BE RD2      38CE
RD3      38DA RD4      38EC Rav      37CB Ret      0528
SETE1    37A1 SUBT     36D4 SUBTR    3715 Set1     37B9
Set2     37C8 Sete     3788 Setlg    37A8 Start    3A2F
Static   3229 Sto      37DF Store    39A8 Store1   39B6
Swap     3757 TEST     371D TMStack  3232 T01      3A21
TURNOFF  39E5 USER     3200 WH0      0C20 WH1      0C24
Warm     0403 addsub   3616 branch   35A6 lic2     359B
lic3     359E lic4     35A2 muldiv   362D opr      356D
oprlic   3569 retn     3768
```

Use Your Computer To Build A Computer

by Jim & Gary Gilbreath

Using a sort program and a program written in Pascal MT$^+$ to help construct a wire-wrap board.

The commercial world of hardware design has tools for logic synthesis, translation of logic equations into TTL logic, timing simulation, PC board layout, wiring list generation, and production control. These tools run on large computers, and cost from $5,000 up for software licenses (mostly *up*).

For the small electronics business and the hobbyist, these tools are more elaborate than necessary, as well as too expensive, since even dial-up time-sharing firms charge more for using these programs than the cost of parts for building the first model.

The tool needed most is an automated way to determine and document *how things are hooked up*—a wire list, in other words. This is useful not only for wrapped-wire construction, but also as an aid for PC board layout and error finding.

We are the authors of Wiremaster, a Z-80 CP/M program which does many things in aid of PC layout and wire-wrap construction. It dawned on us while designing and coding Wiremaster that the most basic help for wrapped-wire and PC board construction could be easily had *without* a special program if an ordinary external file-sorting routine is available. A show and tell session one evening at the local S-100 Innovators group got an appreciative reception, so we decided to write an article about these simple techniques.

We aren't giving away our Wiremaster secrets, but if you have a microcomputer with a disk-based external sort routine, you can use it as a tool to aid you in doing some of the things the industrial world does more automatically. The technique to be described does a lot less than Wiremaster, but can be highly useful in PC layout, error finding, board wiring, and documentation control. Assuming you already have a text editor and a sort program which simply sorts lines into alphabetic order, you won't have to do any programming for the basic method. If you want to get a bit more automatic, you can utilize the program we provide here either directly or by translating into your favorite language.

How To Do It

The first step is to draw the schematic, making sure you have labeled pin numbers for every IC, connector, and discrete component. The microcomputer can't help you much there. Don't bother to draw all the interconnecting lines for bussed components, just be sure you know the name of signal which drives every pin.

Notice that your schematic (and all others, really) is just a pictorial layout of components, and how they are connected. Notice that each component has a name (e.g., IC2) and a type (e.g., 7474).

Each component also has input and output connections which we call pins, since this is the usual case with TTL devices. Each pin has a number, which is usually one or two digits, but which could also be alphabetic, such as found on many connectors.

Consider also that the lines which connect the pins together could be given a name, preferably the name of the signal that drives that particular collection of pins. We use the term *network* to mean all the places that a signal goes. For example, the network named DELAYEDCLK in Figure 1 originates on pin 12 of IC1 and goes to IC4 pin 1, and IC4 pin 4. So, if we have a simple list of signal names and the pins they connect to in the form

```
SIGNALNAME SOCKET PIN      (one pin per line)
SIGNALNAME SOCKET PIN
SIGNALNAME SOCKET PIN
```

and then run this file through our sort program, the result will be that all the pins having the same signal on them will be grouped together and, PRESTO!, we have produced an elementary network list, from which we can wire, error check, and help layout a PCB.

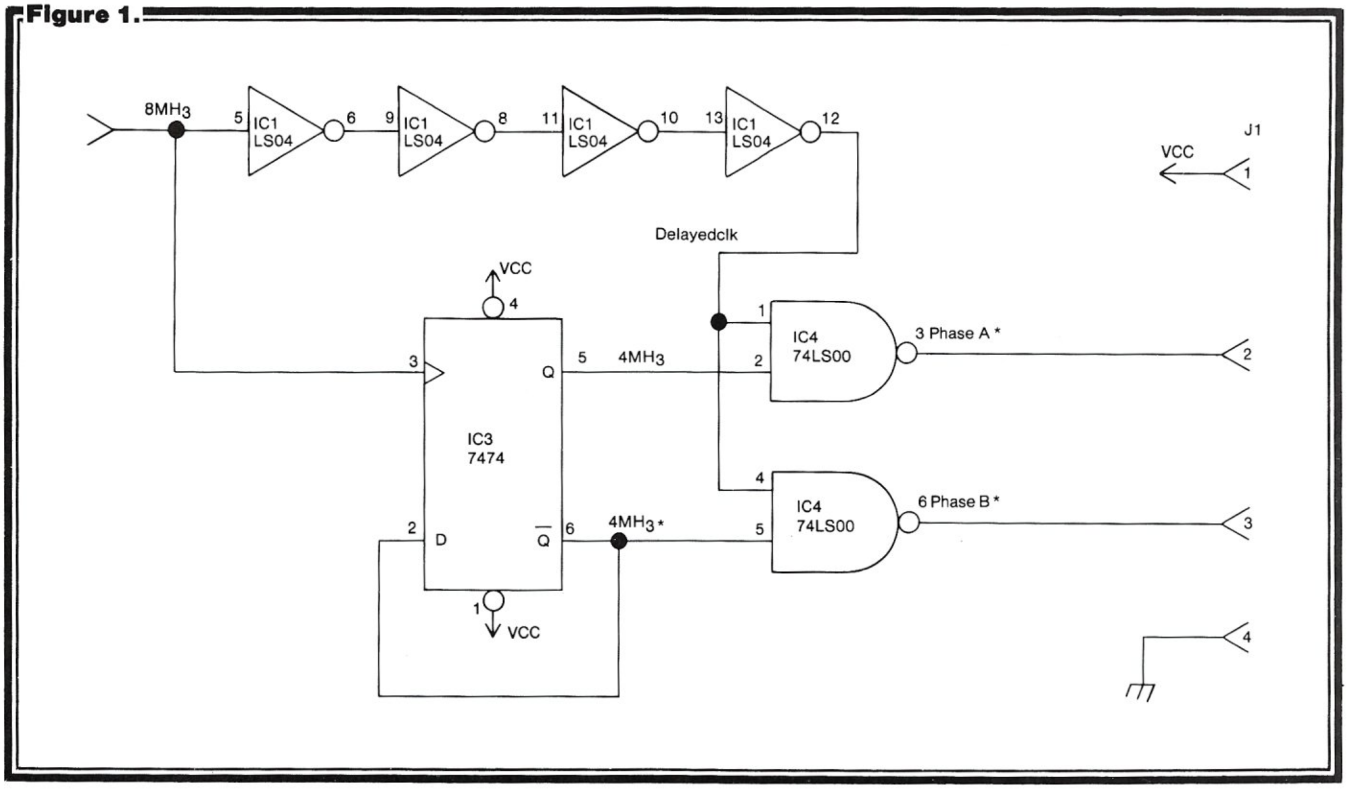
You can then use your screen editor to add separating blank lines, move networks around, and otherwise beautify the output.

Errors can be detected by looking for networks which have only one pin, or places where the same pin has been re-used (perhaps you can write a program to do these for you).

For wire-wrap, you can wire the board directly from this network listing. A wire will go from the first pin to the

Jim & Gary Gilbreath, 7266 Courtney Drive, San Diego, CA 92111.

Figure 1.



second (at level one), another will go from the second pin to the third pin (at level two), the next will go from the third pin to the fourth pin (at level one), etc, until the last wire winds up on the last pin. It is best to do all the level one wires first, then the level two.

For PC layout, the same network listing gives you a picture of all the points on the board that must be connected by track, and provides a checklist to make sure the layout is correct before making that expensive prototype.

Two techniques will be described in detail. The first is used if you have only a sort routine, and the second is used if you have implemented the program given in this article for processing a more convenient input language.

The Simple Method

The first step is to translate the schematic into an input pinlist. This is done by building a text file where every line represents a pin and the name of the signal on it. The format described above is satisfactory, the main thing is that the signal name must be given first so the sort routine can group the signals together.

The easiest way to do this is to get a copy of the schematic you can mark up, and check off each pin as you progress around each IC. Write a line in the above format for each pin (in any order). If you can get someone else to type the information into the computer while you read it off the drawing, it goes quickly.

Signal names should be descriptive of the function being performed, but may be any string of characters at all, just so long as it is not the same as any other signal name in the circuit. If you get tired of inventing names for nondescript little gates, you can always backslide into something like IC5PIN32 as a signal name (if that's the pin that drives the network), but names like LATCH-BIT-5 are more descriptive and much easier to handle later when you want to update the circuit. Table 1 is an input pinlist for the circuit of Figure 1. The sorted output is shown in Table 2.

Table 1: *Input Pinlist*

```
8MHZ .IC1 5
DELA .IC1 6
DELA .IC1 9
DELB .IC1 8
DELB .IC1 11
DELC .IC1 10
DELC .IC1 13
DELAYEDCLK .IC1 12
8MHZ .IC3 3
4MHZ* .IC3 2
4MHZ .IC3 5
4MHZ* .IC3 6
VCC .IC3 1
VCC .IC3 4
DELAYEDCLK .IC4 1
PHASEA* .IC4 3
4MHZ .IC4 2
4MHZ* .IC4 5
DELAYEDCLK .IC4 4
PHASEB* .IC4 6
VCC .J1 1
PHASEA*. J1 2
PHASEB* .J1 3
GND .J1 4
VCC .IC1 14
GND .IC1 7
GND .IC3 7
VCC .IC4 14
GND .IC4 7
```

Table 2: *Output from Sort Program*

```
4MHZ .IC3 5
4MHZ .IC4 2
4MHZ* .IC3 2
4MHZ* .IC3 6
4MHZ* .IC4 5
8MHZ .IC1 5
8MHZ .IC3 3
DELA .IC1 6
DELA .IC1 9
DELAYEDCLK .IC1 12
DELAYEDCLK .IC4 1
DELAYEDCLK .IC4 4
DELB .IC1 11
DELB .IC1 8
DELC .IC1 10
DELC .IC1 13
GND .IC1 7
GND .IC3 7
GND .IC4 7
GND .J1 4
PHASEA* .IC4 3
PHASEA*. J1 2
PHASEB* .IC4 6
PHASEB* .J1 3
VCC .IC1 14
VCC .IC3 1
VCC .IC3 4
VCC .IC4 14
VCC .J1 1
```

The Better Way

You have probably noticed by now (especially if you have tried a sample case) that typing the socket name

Use Your Computer, continued...

over and over agains is a real bore. So let's invent a "language" that makes input a bit easier, and let a program process it to produce a file that the sort program can use as before.

The format we like to use is shown by the example below:

```
.IC3 =74LS74
    1 VCC, 3 8MHZ, 5 4MHZ, 6 4MHZ*,
    4 VCC, 14 VCC, 7 GND,
```

The key points are that pins and signal names are grouped as pairs and separated by commas, that the socket location begins with a period, and that the component type begins with an equal sign. A pinlist in this format for the circuit of Figure 1 is given in Table 3.

A program written in Pascal MT+ is provided that will process a file of this format, and produce three output files containing a list of all the components (.IC), syntax errors encountered (.ERR), and file of parsed output in the form needed for sorting by a simple alphabetic sort program (.PAR). If you don't have Pascal MT+, you can easily translate the program into any of a wide variety of other languages (even Basic).

The program stops far short of the calculated wire lengths, minimization of wiring, error detection, layout aids, and pretty outputs that Wiremaster provides. But it *is* useful, and you can't beat the price. You could write a similar program which takes its input from the sorted file and produces a from-to wiring list and a pictorial network map.

Table 4 shows the parser program's output, consisting of parts list, and pinlist ready for sorting to produce the network list. Table 5 is the final output from the sort program, with annotations to show how the wiring is done for networks with 2, 3, 4, and 5 pins.

Table 3: *Input Language*

```
.IC1 =74LS04
        5 8MHZ, 6 DELA, 9 DELA, 8 DELB, 11 DELB,
        10 DELC, 13 DELC, 12 DELAYEDCLK, 14 VCC,
        7 GND,

.IC3 =7474
        3 8MHZ, 2 4MHZ*, 5 4MHZ, 6 4MHZ*,
        1 VCC, 4 VCC, 14 VCC, 7 GND,

.IC4 =74LS00
        1 DELAYEDCLK, 3 PHASEA*, 2 4MHZ,
        5 4MHZ*, 4 DELAYEDCLK, 6 PHASEB*, 14 VCC,
        7 GND,

.J1 =CONNECTOR
        1 VCC, 2 PHASEA*, 3 PHASEB*, 4 GND,
```

For such a small circuit as this example, the effort may outweigh the benefits. But on projects of ten IC's or more, these techniques will save a great amount of time, reduce errors, and produce a well-documented product.

The Sort Program

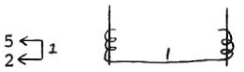
For a circuit of 50 IC's, such as a CPU board or disk controller, the files will be much larger than computer memory, and that's why an "external" sort program is required. If you don't have one and don't want to buy one, we suggest reading *Software Tools* by Kernighan and Plauger (Addison-Wesley, 1976) for a good explanation of how they work and for examples written in nicely structured code. Or send us an eight inch CP/M diskette (with return postage) and we'll give you one of ours. ■

Table 4: *Output from Parser Program*

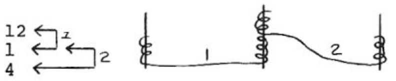
```
Parts List (file.IC)

.IC1=74LS04
.IC3=7474
.IC4=74LS00
.J1=CONNECTOR

Parsed Input (file.PAR)

8MHZ .IC1 5
DELA .IC1 6
DELA .IC1 9
DELB .IC1 8
DELB .IC1 11
DELC .IC1 10
DELC .IC1 13
DELAYEDCLK .IC1 12
VCC .IC1 14
GND .IC1 7
8MHZ .IC3 3
4MHZ* .IC3 2
4MHZ .IC3 5
4MHZ* .IC3 6
VCC .IC3 1
VCC .IC3 4
VCC .IC3 14
GND .IC3 7
DELAYEDCLK .IC4 1
PHASEA* .IC4 3
4MHZ .IC4 2
4MHZ* .IC4 5
DELAYEDCLK .IC4 4
PHASEB* .IC4 6
VCC .IC4 14
GND .IC4 7
VCC .J1 1
PHASEA* .J1 2
PHASEB* .J1 3
GND .J1 4
```

Table 5: *Output from Sort Program (Network Listing)*

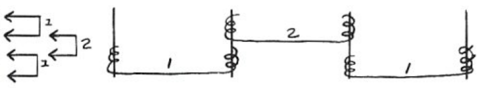
```
4MHZ .IC3 5
4MHZ .IC4 2
```



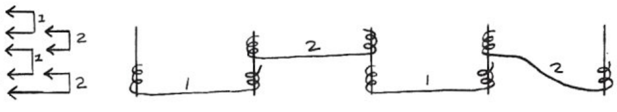
```
4MHZ* .IC3 2
4MHZ* .IC3 6
4MHZ* .IC4 5

8MHZ .IC3 3
8MHz .IC1 5

DELA .IC1 6
DELA .IC1 9

DELAYEDCLK .IC1 12
DELAYEDCLK .IC4 1
DELAYEDCLK .IC4 4
```



```
DELB .IC1 11
DELB .IC1 8

DELC .IC1 10
DELC .IC1 13

GND .IC1 7
GND .IC3 7
GND .IC4 7
GND .J1 4

PHASEA* .IC4 3
PHASEA* .J1 2
PHASEB* .IC4 6
PHASEB* .J1 3
```



```
VCC .IC1 14
VCC .IC3 1
VCC .IC3 4
VCC .IC4 14
VCC .J1 1
```



WIREMASTER

A COMPILER FOR HARDWARE

- **WIREMASTER** is a software tool to aid in the design, layout, and construction of electronic hardware. Although intended primarily for wire wrap, it is also highly useful in the layout, error checking, and trouble-shooting of PC boards.

- Inputs are easily derived directly from the schematic diagram and fed to **WIREMASTER** in a CP/M® text file. Outputs include a **network map** that graphically shows all pins and wires (no plotter required), a **wire list** sorted by lengths and levels, a **parts list**, signal and pin **cross-references**, and **wrap count** and **continuity** checklists which ensure a perfect job.

- **WIREMASTER** checks for syntax errors, wires that go nowhere, and duplicated pins. Network lengths are **minimized**, and wire lengths are calculated and sorted in descending order so that the shorter wires on top hold down the longer wires beneath for a neat wiring job.

- The resulting information is then used for wiring, PC board layouts, error-checking, component stuffing, and system debugging. This makes a complete and easily updated documentation package. Although it runs on small computers, **WIREMASTER** can handle large projects.

- The new **WIREMASTER** Version 3.11 features wire and track minimization, and includes **CHANGEMASTER**, a new program which precisely documents all differences between two versions of a board.

- **CHANGEMASTER** eliminates the confusion and loss of control caused by changes to a board during debugging and modification. It outputs a map of all networks which have been added, deleted, or modified, and automatically generates REMOVE and ADD lists so an old board can easily be brought into exact conformance with the latest design.

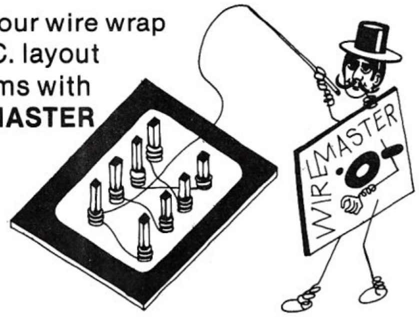
- **WIREMASTER** runs on **any Z80®** CP/M system of 47K or larger, including S-100 systems, TRS-80® Model II, Osborne, and Apple II® with SoftCard® . The diskette also includes a SORT utility, a pagination and multi-column printing program, and **FIX**, a program which finds all bad spots on a disk, tells you which files they are in, places the bad sectors in a single file, and recovers erased files.

WIREMASTER plus **CHANGEMASTER**. . .	$200
Manual only.	$ 15
Update to 3.11 plus **CHANGEMASTER**. .	$ 50
Update to 3.11 only.	$ 15

Calif. residents add 6% sales tax

Checks, CODs, company POs are OK.
NO CHARGE CARDS

Whip your wire wrap and P.C. layout problems with **WIREMASTER**



AFTERTHOUGHT ENGINEERING
7266 Courtney Drive
San Diego, California 92111
(714) 279-2868

```
PROGRAM PARSE;

{---------------------------------------------------------------------}
{ Program to process a CP/M file in the form:                         }
{                                                                     }
{  .SOCKET =ICTYPE                                                    }
{       PIN-NO SIGNAL, PIN-NO SIGNAL,                                 }
{       PIN-NO SIGNAL, ETC                                            }
{                                                                     }
{ If not supplied on the command line, the program asks for the       }
{ input file name.  It then asks for an output base file name, and    }
{ produces 3 output files, named BASE.ERR, BASE.IC, and BASE.PAR,     }
{ containing error messages, parts list, and parsed signal-pin list,  }
{ respectively.  The .PAR file, when sorted into alphabetical order,  }
{ becomes a network list which is useful for wire-wrap, PC layout,    }
{ error checking, and documentation.                                  }
{---------------------------------------------------------------------}

CONST
  CPM_BUF_ADDR = $80;   {address of CP/M command line}

VAR
  ERROR : BOOLEAN;
  RESULT: INTEGER;
  INNAME,                       {file of  input data}
  OUTNAME : STRING;             {base file name for output info}

  CPM_CMD_LINE : STRING[128];   {place to get input file name}

  INFILE : TEXT;                {where data comes from}
  ERRFILE: TEXT;                {where errors go}
  ICFILE  : TEXT;               {where parts list goes}
  PARSEFILE  : TEXT;            {where parsed output goes}

  LINENUM : INTEGER;            {keeps track of line numbers on input file}

  TERM : BYTE;                  {what terminated each word}

  START_OF_WORD,                {chars that words can begin with}
  END_OF_WORD : SET OF CHAR;    {chars that can end words legally}

  WORD  : STRING;               {where getword puts the word it got}
  SOCKET,                       {where socket name goes}
  ICS,                          {string to save socket and ic type}
  PINNAME : STRING;             {so it is}

EXTERNAL PROCEDURE @HLT;        {so can exit easily from the deep}

PROCEDURE INITFILES;
VAR
  P : ^BYTE;
  PROCEDURE ERR_CHK(FILENAME:STRING);
  BEGIN
    IF IORESULT = 255 THEN
      BEGIN
        WRITELN('Unable to open: ',FILENAME);
        @HLT
      END
  END;

BEGIN
  P := CPM_BUF_ADDR;
  MOVE(P^,CPM_CMD_LINE,SIZEOF(CPM_CMD_LINE));      {get what user typed}
  IF LENGTH(CPM_CMD_LINE) <> 0 THEN                {user specified input}
    BEGIN
      WRITELN('Input file is: ',CPM_CMD_LINE);
      INNAME := CPM_CMD_LINE
    END
  ELSE                                             {must ask for input file}
    BEGIN
      WRITE('Input file name? ');
      READLN(INNAME);
      IF LENGTH(INNAME) = 0 THEN
        @HLT;
    END;
  ASSIGN(INFILE,INNAME);
  RESET(INFILE);
  ERR_CHK(INNAME);

  WRITELN;
  WRITE('Base name of output files: ');            {ask for base output file}
  READLN(OUTNAME);
  ASSIGN(ERRFILE,CONCAT(OUTNAME,'.ERR'));          {assign an error file}
  REWRITE(ERRFILE);
  ERR_CHK(CONCAT(OUTNAME,'.ERR'));

  ASSIGN(ICFILE,CONCAT(OUTNAME,'.IC'));            {assign a parts list file}
  REWRITE(ICFILE);
  ERR_CHK(CONCAT(OUTNAME,'.IC'));

  ASSIGN(PARSEFILE,CONCAT(OUTNAME,'.PAR'));        {assign a file for main output}
  REWRITE(PARSEFILE);
  ERR_CHK(CONCAT(OUTNAME,'.PAR'));

END;

PROCEDURE INITSETS;
BEGIN
  START_OF_WORD := [CHR(0)..CHR(255)] - [CHR(13),CHR(9),' '];
  END_OF_WORD   := [CHR(13) , ' ' , CHR(9) , ','];
END;

PROCEDURE CHECK_FOR_EOLN;

BEGIN
  IF EOLN(INFILE) THEN
    LINENUM := LINENUM + 1
END;

{gets next word into global string WORD}
```

```
FUNCTION GETWORD:BYTE;
VAR
 I : INTEGER;
BEGIN
    WHILE (NOT(INFILE^ IN START_OF_WORD)) AND (NOT EOF(INFILE)) DO
      BEGIN
        CHECK_FOR_EOLN;
        GET(INFILE);                {pass by spaces, tabs}
      END;

    IF EOF(INFILE) THEN
        HANDLE_EOF;

  I := 1;                           {assemble the word now}
  REPEAT
    WORD[I] := INFILE^;             {build string}
    I := I + 1;
    GET(INFILE);
    CHECK_FOR_EOLN;
  UNTIL INFILE^ IN END_OF_WORD;

  IF EOLN(INFILE) THEN
    GETWORD := 13                   {return c/r for end of line}
  ELSE
    GETWORD := INFILE^;             {else return termination character}
  GET(INFILE);

  WORD[0] := CHR(I-1);              {set string length byte}

END;


PROCEDURE HANDLE_EOF;

  PROCEDURE CHK_FOR_ERR(name:string);
  BEGIN
    IF IORESULT = 255 THEN
      WRITELN('ERROR in closing ',name);
  END;

BEGIN
  WRITELN;
  WRITELN('Finished');
  WRITELN(ICFILE,ICS);                        {write last of parts list}
  CLOSE(INFILE,RESULT);
  CLOSE(ERRFILE,RESULT);
  CHK_FOR_ERR('Error file');
  CLOSE(ICFILE,RESULT);
  CHK_FOR_ERR('.IC file');
  CLOSE(PARSEFILE,RESULT);
  CHK_FOR_ERR('.PAR file');
  @HLT
END;


BEGIN (* MAIN PROGRAM *)
  LINENUM := 1;
  INITFILES;
  INITSETS;

WHILE TRUE DO
BEGIN
  REPEAT
    TERM := GETWORD;                          {get next word into WORD}
  UNTIL WORD[1] = '.';                        {find first period}

    SOCKET := WORD;                           {save IC socketname}
    ICS := WORD;                              {also in another string}

  REPEAT (* UNTIL ERROR *)
    TERM := GETWORD;                          {read pin name, probably}

    IF WORD[1] = '=' THEN
        ICS := CONCAT(ICS,WORD)               {add IC type to socket string}

    ELSE IF WORD[1] = '.' THEN                {new socket}
      BEGIN
        SOCKET := WORD;                       {save new socket name}
        WRITELN(ICFILE,ICS);
        ICS := WORD;                          {save it here too}
      END
    ELSE
      BEGIN
        IF WORD[1] = '$' THEN                 {ignore any word beginning with $}
           TERM := GETWORD;
        IF NOT (TERM IN [CHR(9),      CHR(13)]) THEN
          ERROR := TRUE                       {pin didnt end in tab, space or c/r}
        ELSE
          BEGIN
            PINNAME := WORD;                  {save pin name}
            TERM := GETWORD;                  {read signal name}
            IF NOT (TERM IN [',', CHR(13)]) THEN
              ERROR := TRUE                   {signal name must end in comma or c/r}
            ELSE
              {output completed line of signal, socket, pin}
              WRITELN(PARSEFILE,WORD,' ',SOCKET,' ',PINNAME)
          END


          END
      UNTIL ERROR;
      WRITELN('Error on line ',LINENUM);
      WRITELN(ERRFILE,'Error on line ',LINENUM);
      ERROR := FALSE                                {set back to normal for next try}
    END;

    END.
```

Plain Talk About Business Computers

Can a Small Computer Really Save You Time?

Time is Money

Theophrastus said time was the most valuable thing a man could spend. Fifteen centuries later Haliburton agreed saying, "we reckon hours and minutes to be dollars and cents." Today, time is more valuable than ever—and more fleeting.

About the only way to gain time is to use it more efficiently and effectively. That's where we come in.

Small Business Computers—by the way, the "small" refers to computers, not to business—will dramatically increase your effectiveness and help save you time and money. How so?

You get flagrantly honest evaluations and reviews of computers and software. We don't just tell you what a program can do; we tell you what it doesn't do, what it does poorly, and what it should do for the price. If advertisers don't like that, we don't want their business, and you're better off without them. Fortunately, most companies appreciate our honesty. In fact, one of our reviewers has gained a reputation because of the many software houses that have incorporated his suggestions into their products. We're proud of that.

Photo courtesy of Alanthus Data Communications Corp.

Plain Talk

Small Business Computers explains the complexity of today's computerized business world without the technical jargon and doubletalk that may have held you back before. In its easily comprehensible "how-to" style, *Small Business Computers* answers your questions while providing the information you need to make some tough decisions. As you select, purchase, and install your computer system, *Small Business Computers* will guide you through each step calmly and comfortably—helping you to evaluate your computer needs and avoid unnecessary pitfalls. As you use your computer, be it mini or micro, *Small Business Computers* will be there to help you do so efficiently and with confidence while informing you of the latest developments and future possibilities of computers in business.

For Example

You have just purchased a mailing list program. Everything is fine until the file has to be sorted by zip code. If the program has that capability, all is well. If not, you have a big problem. If you had just invested a few hours reading *Small Business Computers*, you would have known what functions to look for before buying the program; you would have known how to plan for future needs. That's just one example. Expand this concept into other areas, other programs and systems, and you can see what you get for your investment.

Added Expertise

As the newest member of the Creative Computing family of fine computer publications, *Small Business Computers* will be expanding to offer subscribers more valuable information than ever before. *Creative Computing* editors and contributors will be unleashing their business expertise in *Small Business Computers* through articles, evaluations and applications of particular interest to the business person. *Creative Computing* has a reputation of editorial excellence and integrity built on unbiased, in-depth product evaluations; articles by top thinkers in the field; and pragmatic, innovative applications.

One management consulting firm, for example, used the Shell-Metzner sort described in *Creative*, and saved $3000 a month, and we still receive letters thanking us for the hardhitting, candid, evaluation of word processing printers we published over a year ago, and which, incidentally, cost us several advertisers.

All this knowledge and experience will now be available to business people in *Small Business Computers*.

So, don't let anyone give you that old story about how complicated and difficult computers are. We don't buy that. Our magazine—our whole philosophy—revolves around the sharing of honest information. If you don't know where to start, we'll put you on the right track. If you're already on the road, we'll show you the best route.

For Any Size Business

Whatever your business—manufacturing or banking, retail or research—*Small Business Computers* will increase your efficiency and help save you time and money.

Subscribe today; *Small Business Computers* is the best consultant your business will ever have.

Order Today

To order your subscription to *Small Business Computers* send $16.00 for 1 year (6 issues). If you prefer, call our toll free number *800-631-8112* (in N.J. 201-540-0445) to put your subscription on your MasterCard, Visa, or American Express card. Canadian and other foreign surface subscriptions are $21.00 per year and must be pre-paid. We guarantee that you will be completely satisfied or we will refund the remaining portion of your subscription.

Send orders to:

Small
Business Computers
Magazine

39 E. Hanover Ave.
Morris Plains, NJ 07950
800-631-8112
(In NJ 201-540-0445)

DISKINDX File Reorganization Program

by R.W. Jenks

A disk index reorganization program for North Star users

Users of the North Star MDS and Horizon are soon pleased with the power of the software provided. The system software, including the popular North Star Basic, offers many features which are easy to use due to structural simplicity. Though a detailed understanding of the workings requires a careful and involved inspection of the disassembled machine language code, enough information is provided by North Star about the underlying structure to give the user a significant capability for direct access and modification.

This article describes a Basic program to provide the function of a disk index reorganizer, a useful utility not provided in the system software.

A Brief Review of the North Star Microdisk System

The North Star floppy disk system is composed of from one to three Shugart SA400 single-sided Minifloppy disk drives, or from one to four SA450 double-sided drives (or compatible equivalents), the disk controller board, Disk Operating System (DOS) and utilities. The 5.25" media is hard-sectored to ten sectors per track, 35 tracks per side with one block of 256 bytes per sector in single density and two blocks (512 bytes) per sector in double density.

The DOS may be accessed either directly through disk commands and machine language calls to disk routines, or indirectly through utility programs and North Star Basic file handling statements (refer to Table 1).

Files are implemented by first testing a diskette for the absence of hard errors, and then initializing the diskette to a standard format as follows: for every track, approximately 96 microseconds after each sector pulse is detected, the DOS writes a preamble of 16 null bytes (00H), followed by a sync byte (0FBH), either 256 or 512 data bytes of ASCII space (20H) and a cyclic check byte. Files may now be created as index entries with the data space measured in units of blocks, but allocated as an integer number of sectors. The index table takes the form of an assumed, unnamed special file preceeding all other files. North Star allocates the first four sectors as the index.

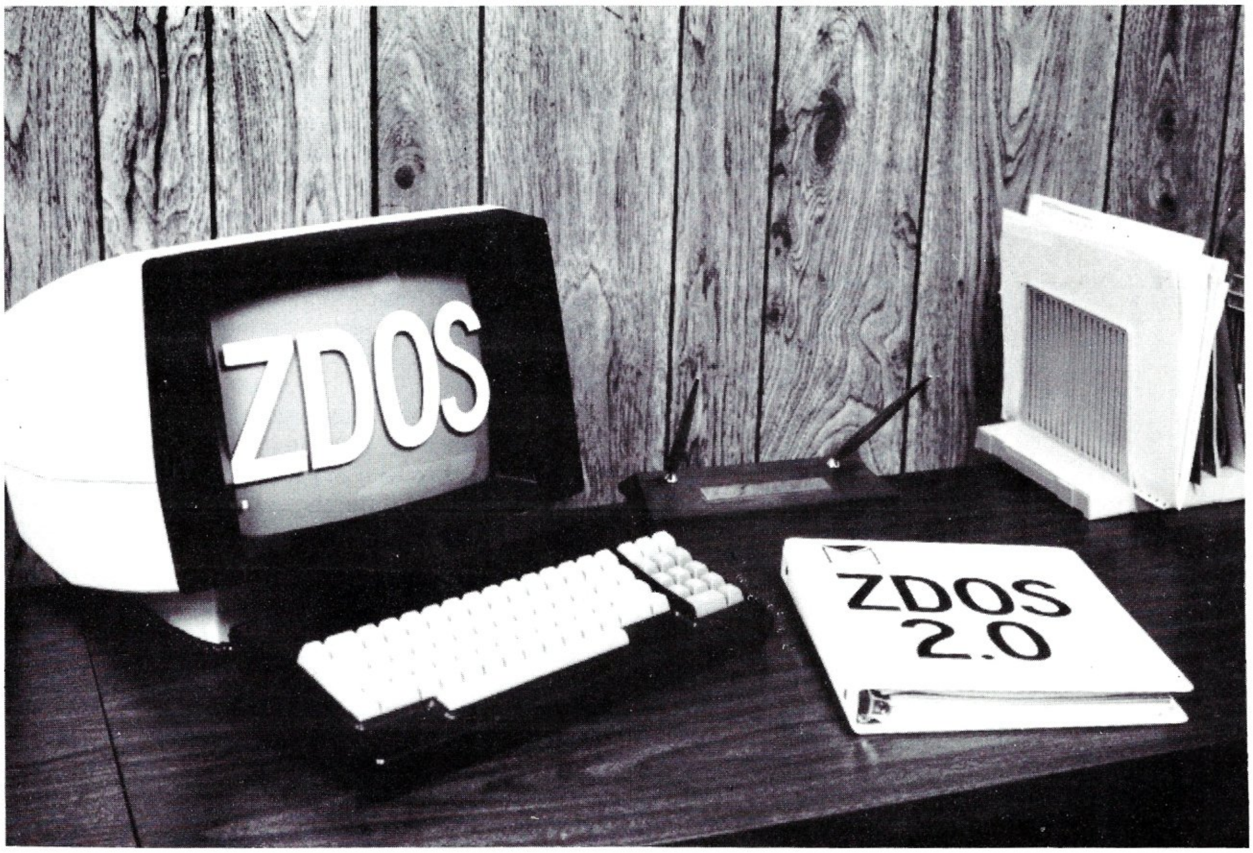
Each index entry is a 16 bytes long, describing one autonomous file. Overlapping files can be created which, though usable for certain purposes, may cause problems. The entry format consists of, in order, eight characters for a name, two bytes in machine language format for the address of the starting disk block, two bytes for the file length in blocks, one byte as a file type/density indicator and three bytes for type dependent information. The name may have any printable characters except a comma or imbedded space. A file may be as long as the number of blocks on a disk, less those allocated for the index. The type byte reserves bit 7 set as a double

Table 1:
Means of accessing diskettes through the North Star operating system.

CODE	FUNCTION	OPERANDS (some optional)
	DOS COMMANDS	
LI	List diskette index	Output device, drive #
CR	Create a disk file	Name, length, start address, density
DE	Delete an index entry	Name
TY	Set file type	Name, type, start address
GO	Load and run machine language program	Name
LF	Load file to memory	Name, memory address
SF	Save a file from memory	Name, memory address
RD	Read blocks from disk	Disk address, memory address # of blocks, density
WR	Write blocks to disk	Disk address, memory address # of blocks, density
IN	Initialize a diskette	Drive #, density
	DOS UTILITIES	
DT	Disk test	Drive #
CF	Copy file	Source file name, destination file name
CD	Copy diskette	Source drive #, distination drive #
CO COMPACT	Compact diskette file space	Drive #, density
	MACHINE LANGUAGE ACCESSIBLE DISK ROUTINES	
DLOOK	Search index for file name	Default drive #, pointer to file name
DWRIT	Write directory entry to index	(Must follow DLOOK)
DCOM	Command disk activity	# of blocks, command, drive #, density, starting memory address, starting disk address
LIST	List disk index	Drive #, output device
	DISK ACCESS THROUGH NORTH STAR BASIC	
LOAD	Load BASIC program	Name
SAVE	Save BASIC program	Name
NSAVE	Save BASIC program as a new file	Name
CAT	Catalog, list disk index	Output device, drive #
CREATE	Create a disk file	Name, length, type
DESTROY	Delete a disk file	Name
FILE	Determine type/existence of disk file	Name
OPEN	Open a disk file for access	File number, type, name, size
CLOSE	Close a disk file to further access	File number
READ	Read data from a file	File number, pointer, variable list
WRITE	Write data to a file	File number, pointer, data list
TYP	Return the type of the next data element in the file	File number

R.W. Jenks, P.O. Box 962, Islamorada, FL 33036.

STILL TIED TO A HOBBYIST ORIENTED O/S ?
UTILIZE THE FULL POTENTIAL OF YOUR Z-80 BASED SYSTEM
WHILE MAINTAINING COMPATIBILITY WITH CP/M *



NEW FROM MICRO-LOGIC
ZDOS™ 2.0

- More power than CP/M* without losing additional user memory.
- Full software compatibility with CP/M*.
- Supports floppy or hard disks.
- Up to 60% faster Disk I/O.
- Both maskable and non-maskable interrupt capability.
- Designed for networking or stand alone installations.
- Directory linking to other user numbers.
- Fully symbionic printer spooling.
- Unlimited nesting of SUBMIT files.
- Easy upgrade from CP/M* - use your present BIOS if desired.
- Clear, concise user reference manual.
- Full user support service for registered owners.
- Affordable Price.

MICRO-LOGIC

MICRO-LOGIC INC.
P.O. Box 4324
Hamden, CT 06514

ZDOS is a trademark of MICRO-LOGIC INC.
* CP/M is a trademark of Digital Research.

MICRO-LOGIC

MICRO-LOGIC INC
P.O. Box 4324
Hamden, CT 06514

□ Please send further information on ZDOS.
□ Send me ZDOS (manual incl.) Enclosed is my check for $149.95.
□ Send ZDOS Manual only ($49.95).

NAME: ______________________________

TITLE: ______________________________

COMPANY: ______________________________

ADDRESS: ______________________________

____

PURCHASING STATUS: □ AUTHORIZE □ RECOMMEND

MY CURRENT SYSTEM 'CONFIGUATION:

HARDWARE: ______________________________

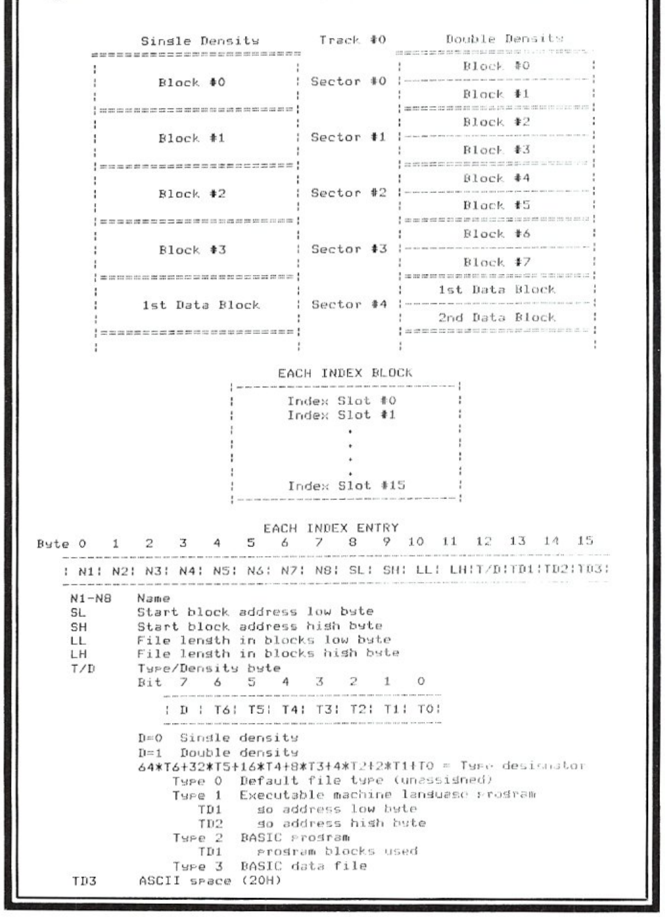
____

OPERATING SYSTEM: ______________________________

density flag. The remaining bits may define any of 128 different file types, although four have predefined meanings (see Figure 1).

A file is created by placing a valid file entry into the index. Certain operations are assumed by the DOS. First, if any unused slots are free the index entry will be assigned to the first available slot. Secondly, disk space for the data blocks of the file will be allocated after the last disk block currently assigned to a file.

Figure 1: *North Star disk index format.*



```
   Single Density          Track #0        Double Density
=========================              ==========================
:                         :            :        Block #0         :
:        Block #0         : Sector #0  :-------------------------:
:                         :            :        Block #1         :
:=========================:            :=========================:
:                         :            :        Block #2         :
:        Block #1         : Sector #1  :-------------------------:
:                         :            :        Block #3         :
:=========================:            :=========================:
:                         :            :        Block #4         :
:        Block #2         : Sector #2  :-------------------------:
:                         :            :        Block #5         :
:=========================:            :=========================:
:                         :            :        Block #6         :
:        Block #3         : Sector #3  :-------------------------:
:                         :            :        Block #7         :
:=========================:            :=========================:
:                         :            :     1st Data Block      :
:     1st Data Block      : Sector #4  :-------------------------:
:                         :            :     2nd Data Block      :
:=========================:            :=========================:
:                         :            :                         :

                      EACH INDEX BLOCK
                 :-------------------------:
                 :      Index Slot #0      :
                 :      Index Slot #1      :
                 :            .            :
                 :            .            :
                 :            .            :
                 :            .            :
                 :      Index Slot #15     :
                 :-------------------------:

                       EACH INDEX ENTRY
Byte 0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
   ------------------------------------------------------------------
   : N1: N2: N3: N4: N5: N6: N7: N8: SL: SH: LL: LH:T/D:TD1:TD2:TD3:
   ------------------------------------------------------------------
   N1-N8    Name
   SL       Start block address low byte
   SH       Start block address high byte
   LL       File length in blocks low byte
   LH       File length in blocks high byte
   T/D      Type/Density byte
            Bit  7   6   5   4   3   2   1   0
                ---------------------------------
                : D : T6: T5: T4: T3: T2: T1: T0:
                ---------------------------------
            D=0  Single density
            D=1  Double density
            64*T6+32*T5+16*T4+8*T3+4*T2+2*T1+T0 = Type designator
                 Type 0  Default file type (unassigned)
                 Type 1  Executable machine language program
                     TD1    go address low byte
                     TD2    go address high byte
                 Type 2  BASIC program
                     TD1    program blocks used
                 Type 3  BASIC data file
   TD3      ASCII space (20H)
```

This brings up difficulties which leads to the purpose of DISKINDX.

DISKINDX: File Reorganization

All is fine when files are set up sequentially. The first index slot takes the first file name entry and the first disk blocks after the index are assigned as data space. Following files are assigned to both successive index slots and disk blocks. However, disk use eventually leads to file deletions and new file creations. When a file is deleted, its blocks are temporarily lost as available storage if they were not the last assigned on the disk. The index slot, on the other hand, becomes free for a new entry. As new files are tacked on, eventually the DOS is unable to find space for another file. At this stage disorder prevails. There is usually little correlation between the order of a file name in the index and its starting disk address; the order of files in the index does not correspond to the order in which files were set up on the disk; there are gaps of unassigned and unused blocks scattered throughout.

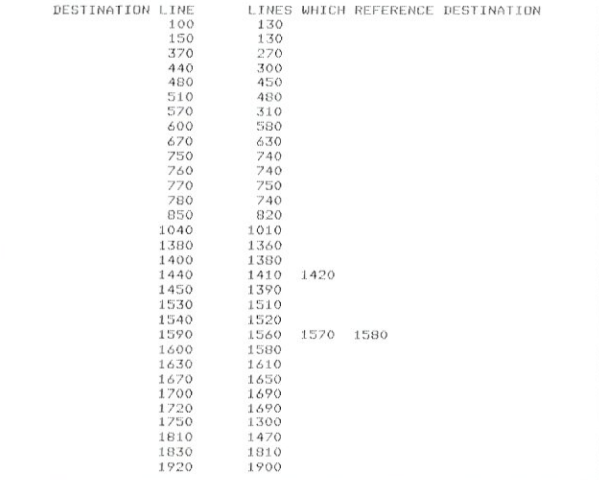
File space can be easily recovered through use of the COMPACT utility supplied by North Star. This utility recovers disk blocks by moving allocated blocks towards the front of the disk while keeping track of new starting block addresses and modifying the index entries accordingly. However, this utility will not recover all the unused blocks and does not restore order to the index. The data blocks allocated for a Basic program may exceed the actual blocks holding program code. Whenever a file is NSAVED, the Basic interpreter allocates three extra sectors for program expansion. Since the disk index format keeps track of superfluous program blocks, these blocks may be recovered.

DISKINDX has the following features: an ability to recover unused Basic program blocks through modification of the file length bytes of the index entry; an indication of available and recoverable disk blocks and location; an indication of overlapping files; an easy means of examing all 16 bytes of an index entry; the ability to reorganize the disk index such that the index entries correspond in increasing order with the actual file starting blocks, and thus may also indicate correctly the order in which files have been created on the disk.

The Program

Listing 1 is the DISKINDX program written in Release 5.2 North Star Basic. The program is not optimized for size and requires over 7000 bytes for operation (single density). Size was not considered a limitation.

Table 2: *A correlation of entry and branch line numbers.*



DESTINATION LINE	LINES WHICH REFERENCE DESTINATION		
100	130		
150	130		
370	270		
440	300		
480	450		
510	480		
570	310		
600	580		
670	630		
750	740		
760	740		
770	750		
780	740		
850	820		
1040	1010		
1380	1360		
1400	1380		
1440	1410	1420	
1450	1390		
1530	1510		
1540	1520		
1590	1560	1570	1580
1600	1580		
1630	1610		
1670	1650		
1700	1690		
1720	1690		
1750	1300		
1810	1470		
1830	1810		
1920	1900		

The program contains some general purpose outlines for implementing direct user modification of disk file index entries which the reader might have use of for other purposes (such as alphabetizing). The description in Listing 2 of the labels used should help in understanding the program, which is structured in a loose sort of way through function and subroutine calls. Scattered REM statements and the entry point and branch line numbers in Table 2 will aid the reader if drawing up a flow chart would be helpful. The program can be easily modified to support a backup index on the same or another disk.

A word about the output select function. This routine modifies the DOS output routines as a means of selecting an outut device. The program should not be run without either substituting a RETURN which will disable this function, or determining that the actions it will take are compatible with your system. An alternate method is to add device designators to the applicable print statements, as featured in North Star Basic, and modifying the routine to change the device designator rather than the software.

> *DISKINDX has the following features: an ability to recover unused Basic program blocks through modification of the file length bytes of the index entry; an indication of available and recoverable disk blocks and location; an indication of overlapping files, and much more.*

Also, run this program initially with a dummy index on an unused disk until you are satisfied that the program has been copied correctly, that it works reliably and as expected.

Before the program will run, diskettes must have a type 3 Basic data file created on them called INDEX covering the disk index blocks. This is done by using the optional starting block parameter in the CR command, and causes no problems other than the possible inconvenience of taking one index slot. Modify the constants in line 30 if your system is other than single density and a 24 line terminal (0 disables the line count for printer use).

A diskette may optionally be given a name in the form of a file entry of type 4 (modify the program to change the type number recognized as a disk name). This will, however, reduce by one the number of data files which may be created on the disk, but the ability to call up a file containing disk use information (as a text file) proves useful in particular applications. One example is a diskette devoted to 16 (or 31) twenty-one (22) block long pages of text, where the disk use file briefly describes the contents of each page.

Operation

A sample dialog with the program is shown in Listing 3. A carriage return alone in response to a prompt bypasses without action. The program will select drive one in default. Output waits for viewing after a full screen of data is printed. The program makes a first pass to display the index, shows block use information, reorders the entry list and prepares to write a new index when authorized. If there are excess Basic program blocks which may be recovered, the program will ask if this should be done. Be advised that a positive response will shorten the allocation of *every* program on the disk to recover the maximum number blocks.

Remarks

I have found this program to be useful for a variety of purposes, especially in keeping track of a large number of files. When creating new files on diskettes reserved for data it is desirable to have the new file entry appear at the end of the index so that it does not have to be found by looking through 60-odd other entries, and also to know the order in which the files were entered, as when they are modifications of data in previous files. I found that it is preferable to initially allocate a large amount of disk space when programming to prevent the wasted time of correcting a bad save after additions are made. When the development work is done, recovering the unused blocks is an easy matter of using DISKINDX followed by the COMPACT utility.

The additions in North Star Release 5.2 were welcome, and I expect added software support will continue. Perhaps North Star would even see merit in using the ideas of DISKINDX as a basis for another machine language utility. But why wait? Put DISKINDX in service now.

Reference

North Star *System Software Manual,* SOFT-DOC Revision 2.1, North Star Computers, Inc., 14440 Catalina St., San Leandro, CA 94577.

Listing 1: *North Star disk index reorganizing program.*

```
10   REM  "DISKINDX"  BY R W JENKS  10/25/79   MOD 9/11/81
20   REM  FOR NORTH STAR SOFTWARE RELEASE 5.2 AT 100H
30 D1=2 \L1=24   \REM  DENSITY, TERMINAL LINES
40 LINE 79
50 DIM B1(15),E1$(60),S1(64*D1,2),B1$(16*64*D1)
60 D=FNO1("T")
70 ! CHR$(27),"*",CHR$(13),    \REM  CLEAR SCREEN
   SOROC IQ-120 (19200 BAUD)
80 INPUT "PRINTOUT (Y)?",R1$  \D=FNL(1)
90 IF R1$="Y" THEN D=FNO1("P")
100 INPUT "DRIVE #:",D1$ \D=FNL(1)
110 IF D1$="" THEN D1$="1"
120 I1$="INDEX,"+D1$
130 IF FILE(I1$)=3 THEN 150 \! I1$," NOT ON DISK" \GOTO 100
140         REM  **** MAIN
150 D=FNG2("N")
160 S1$=""
170 IF F1=1 THEN INPUT "SHORTEN ALL PROGRAMS TO MINIMUM
    SIZE (S)?",S1$
180 D=FNL(1)
190 IF S1$="S" THEN D=FNG2(S1$)
200 END
210         REM  ##### FUNCTION
220         REM  *** ACCESS WHOLE INDEX, REORDER, SHORTEN
230 DEF FNG2(M1$)
240 OPEN #1,I1$
```

```
250 F1=0 \S1=0 \S2=0 \S3=0
260  FOR B1=0 TO 4*D1-1
270  GOSUB 370
280  D=FNG1(M1$)
290  NEXT
300 GOSUB 440
310 GOSUB 570
320 CLOSE #1
330 RETURN 0
340 FN END
350         REM  ### SUBROUTINE
360         REM  * BLOCK HEADING
370 ! \D=FNL(1)
380 ! "BLOCK",%2I,B1,": ENTRY  LABEL",
390 ! "     ADDR    # SCTRS  TYPE   Type Depn't Info
    Excess Sctrs"
400 D=FNL(1)
410 RETURN
420         REM  ### SUBROUTINE
430         REM  * BLOCK USE DATA
440 ! %# \D=FNL(1)
450 IF S1<=S2 THEN 480
460 ! "WILL RECOVER",S1-S2," SECTOR", \D=FNS1(S1-S2) \!
    " BY COMPACTING DISK"
470 D=FNL(1)
480 IF S3=0 THEN 510
```

```
490 ! "WILL RECOVER",S3," SECTOR", \D=FNS1(S3) \! " BY SHORTENING PROGRAMS"
500 D=FNL(1) \F1=1
510 ! 350-S1," SECTOR", \D=FNS1(350-S1) \! " CURRENTLY AVAILABLE FOR STORAGE"
520 D=FNL(1)
530 ! \D=FNL(1)
540 RETURN
550        REM  ### SUBROUTINE
560        REM  ** REORDER INDEX
570 ! "* ORDERING IN PROGRESS" \D=FNL(1)
580  FOR X1=1 TO 64*D1 \IF S1(X1,0)=0 THEN EXIT 600 \NEXT
590        REM  * SORT/EXCHANGE
600 X1=X1-1
610  FOR X2=1 TO X1-1
620   FOR X3=X2+1 TO X1
630   IF S1(X3,1)>=S1(X2,1) THEN 670
640    FOR X4=0 TO 2
650    S1(0,X4)=S1(X2,X4) \S1(X2,X4)=S1(X3,X4) \S1(X3,X4)=S1(0,X4)
660    NEXT
670   NEXT
680  NEXT
690        REM  * DISPLAY ORDER, UNUSED/OVERLAPPING SECTORS
700 S1(0,1)=0 \S1(0,2)=0
710  FOR X3=1 TO X1
720  ! %6I,S1(X3,0),S1(X3,1),
730  D2=S1(0,1)+S1(0,2)-S1(X3,1)
740  IF D2>0 THEN 750 \IF D2<0 THEN 760 \! \GOTO 780
750  ! "  OVERLAPPED BY",D2, \GOTO 770
760  ! "  PRECEEDED BY A GAP OF",-D2,
770  ! " SECTOR", \D=FNS1(D2) \!
780  D=FNL(1)
790  S1(0,1)=S1(X3,1) \S1(0,2)=S1(X3,2)
800  NEXT
810        REM  * CHECK IF IN ORDER
820  FOR X3=1 TO X1 \IF S1(X3,0)<>X3 THEN EXIT 850 \NEXT
830 ! "INDEX CLOSE PACKED & IN ORDER" \D=FNL(1) \RETURN
840        REM  * FILL INDEX BUFFER
850  FOR X3=1 TO X1
860  E1=16*(S1(X3,0)-1)
870  P1=16*(X3-1)+1
880   FOR X2=0 TO 15
890   READ #1 %E1+X2,&B2
900   B1$(P1+X2,P1+X2)=CHR$(B2)
910   NEXT
920  NEXT
930        REM  * WRITE INDEX BUFFER
940 INPUT "WRITE NEW INDEX (Y)?",R1$ \D=FNL(1)
950 IF R1$<>"Y" THEN RETURN
960  FOR X3=1 TO P1+X2-1
970  WRITE #1 %X3-1,&ASC(B1$(X3,X3)),NOENDMARK
980  NEXT
990        REM  * CLEAN UP END OF INDEX
1000  FOR X3=1 TO X1
1010  IF S1(X3,0)<=X1 THEN 1040
1020  E1=16*(S1(X3,0)-1)
1030   FOR X2=0 TO 7 \WRITE #1 %E1+X2,&32,NOENDMARK \NEXT
1040  NEXT
1050 ! "INDEX REORDERED" \D=FNL(1)
1060 RETURN
1070        REM  ##### FUNCTION
1080        REM  *** LINES PRINTED
1090 DEF FNL(L2)
1100 IF L1=0 THEN RETURN 0
1110 L3=L3+L2 \IF L3<L1-1 THEN RETURN L3
1120 L3=0 \INPUT "* 'RET' TO CONTINUE",R2$
1130 RETURN 0
1140 FN END
1150        REM  ##### FUNCTION
1160        REM  *** SUFFIX 'S'
1170 DEF FNS1(D3)
1180 IF ABS(D3)<>1 THEN ! "S", \RETURN D3
1190 FN END
1200        REM  ##### FUNCTION
```

```
1210        REM  *** GET 1 BLOCK OF ENTRIES
1220 DEF FNG1(M1$)
1230 ! %#6I,
1240 FOR E2=1 TO 16
1250        REM  * FILL ENTRY BUFFER
1260 E3=16*B1+E2 \E1$=""
1270  FOR X1=0 TO 7 \READ #1,&B1(X1) \E1$=E1$+CHR$(B1(X1)) \NEXT
1280  FOR X1=8 TO 15 \READ #1,&B1(X1) \NEXT
1290        REM  * SKIP EMPTY ENTRY
1300 IF E1$(1,1)=" " THEN 1750
1310        REM  * DECODE PARAMETERS
1320 F2=0
1330 N1=256*B1(9)+B1(8)
1340 N2=256*B1(11)+B1(10)
1350 N3=B1(12)-128*(D1-1)
1360 IF N3>=0 AND N3<=127 THEN 1380 \F2=1
1370 N3=N3-SGN(N3)*128
1380 IF N3<>1 THEN 1400
1390 N4=256*B1(14)+B1(13) \N5=0 \GOTO 1450
1400 N4=B1(13)
1410 IF N3<>2 THEN 1440
1420 IF NOT(D1=2 AND NOT F2 OR D1=1 AND F2) THEN 1440
1430 N4=N4/2 \IF N4<>INT(N4) THEN N4=INT(N4+1)
1440 N5=B1(14)
1450 N6=B1(15)
1460        REM  * BUILD UP REORDER TABLE/SHORTEN ALLOCATION
1470 IF M1$="S" THEN IF N3=2 THEN GOSUB 1810
1480 P1=P1+1 \S1(P1,0)=E3 \S1(P1,1)=N1 \S1(P1,2)=N2
1490        REM  * BUILD UP ENTRY DATA STRING
1500 E1$=STR$(E3)+"  :"+E1$+","+STR$(N1)+":"+STR$(N2)
1510 IF NOT F2 THEN 1530
1520 IF D1=1 THEN E1$=E1$+"D" ELSE E1$=E1$+"S" \! %#5I, \GOTO 1540
1530 IF D1=2 THEN E1$=E1$+"D" ELSE E1$=E1$+"S" \! %#5I,
1540 E1$=E1$+":"+STR$(N3)+":"
1550 ! %#6I,
1560 IF N3<>2 THEN 1590
1570 IF D1=2 THEN IF NOT F2 THEN E1$=E1$+STR$(N4*D1) ELSE 1590
1580 IF D1=1 THEN IF F2 THEN E1$=E1$+STR$(N4*D1) ELSE 1590 \GOTO 1600
1590 E1$=E1$+STR$(N4)
1600 E1$=E1$+":"
1610 IF N3=1 THEN 1630
1620 E1$=E1$+STR$(N5)+":"
1630 E1$=E1$+STR$(N6)+";"
1640        REM  * PRINT ENTRY INFO
1650 IF N3<>4 THEN 1670   \REM  TYPE 4 = DISK NAME
1660 N1$=E1$(1,8) \!"NAME:",
1670 ! TAB(7),E1$,
1680        REM  * ACQUIRE BLOCK USE INFO
1690 IF N3=2 THEN 1700 \! \GOTO 1720
1700 S3=S3+N2-N4 \IF N2<N4 THEN ! "USE ERROR",
1710 IF N2>N4 THEN ! N2-N4 ELSE !
1720 D=FNL(1)
1730 S2=S2+N2
1740 IF N1+N2>S1 THEN S1=N1+N2
1750 NEXT
1760 ! %#,
1770 RETURN 0
1780 FN END
1790        REM  ### SUBROUTINE
1800        REM  ** SHORTEN PROGRAM ALLOCATION
1810 IF N4<N2 THEN 1830 \IF N4=N2 THEN RETURN
1820 ! "CHECK ENTRY FOR ERROR" \D=FNL(1) \RETURN
1830 E1=E3-1 \WRITE #1 %16*E1+10,&N4,NOENDMARK
1840 READ #1 %16*E1+15,&D   \REM  ADVANCE FILE POINTER
1850 N2=N4
1860 RETURN
1870        REM  ##### FUNCTION
1880        REM  *** OUTPUT SELECT  DOUBLE DENSITY DOS
1890 DEF FNO1(O1$)
1900 IF O1$="T" THEN 1920 \IF O1$<>"P" THEN RETURN 0
1910 FILL 2673,5 \FILL 2793,4 \RETURN 1   \REM  OUTPUT TO PRINTER
1920 FILL 2673,3 \FILL 2793,2 \RETURN 0   \REM  OUTPUT TO VDT
1930 FN END
```

Listing 2: *The variables used in DISKINDX.*

B1	Block count
B2	Byte
D	Dummy function return value
D1	Disk density
D2	Block count difference
D3	FNS1 input variable, difference/count
E1	Start of entry pointer
E2	Entry count
E3	Entry order
F1	Excess blocks flag
F2	Different density flag
L1	Terminal lines
L2	Additional lines displayed
L3	Total screen lines displayed
N1	File starting block address
N2	Number of file blocks
N3	File type/density
N4	Go address or actual program blocks
N5	Type dependent information
N6	Type dependent information
P1	String pointer
S1	Last used block
S2	Sum of allocated blocks
S3	Sum of excess blocks
X1	Sort pointer
X2	Loop variable
X3	Loop variable
X4	Array pointer for exchange
B1()	Entry buffer
S1()	Order/block-use array
B1$	Index buffer
D1$	Drive designator
E1$	Entry data string
I1$	Index file designator
M1$	Mode character for shorten program allocation
N1$	Disk name
O1$	Output device
R1$	Response to query
R2$	Response to hold screen
S1$	Shorten allocation flag
FNG1	Handle one block of index entries
FNG2	Handle whole index
FNL	Keep track of lines printed
FNO1	Output select
FNS1	Add pleurality suffix

Listing 3: *An example of DISKINDX in use.*

```
*
PRINTOUT (Y)?
DRIVE #:2
BLOCK 0: ENTRY  LABEL      ADDR    # BLKS    TYPE    Type Depn't Info    Excess Blks
             1  :INDEX    ,     0:      4:      3:      4:     32:     32;
             2  :DOS      ,     4:     10:      0:     32:     32:     32;
             3  :BASIC    ,    14:     50:      1:  10752:     32;
             4  :BRANDEST,     64:     70:      3:     32:     32:     32;
             5  :#CODE    ,   165:      5:      2:      2:     32:     32;       3
NAME:        6  :P.BASIC  ,   134:      1:      4:      1:     32:     32;
             7  :ERRORTRP,    170:     10:      2:      4:     32:     32;       6
             8  :D.BASC-T,    180:     17:      2:     15:     32:     32;       2
             9  :DECODE   ,   135:      8:      2:      5:     32:     32;       3
            10  :ADTABL   ,   143:     22:      2:     19:     32:     32;       3
            11  :LABLTABL,    197:     42:      2:     39:     32:     32;       3
            12  :FLOWCHT  ,   326:     14:      2:     14:     32:     32;
            13  :DATALT   ,   266:     60:      3:     60:     32:     32;
            14  :SNGLSTEP,    347:      2:      1:   6912:     32;
BLOCK 1: ENTRY  LABEL      ADDR    # BLKS    TYPE    Type Depn't Info    Excess Blks
            18  :ADDA     ,   344:      3:      2:      3:     32:     32;
BLOCK 2: ENTRY  LABEL      ADDR    # BLKS    TYPE    Type Depn't Info    Excess Blks
* 'RET' TO CONTINUE
BLOCK 3: ENTRY  LABEL      ADDR    # BLKS    TYPE    Type Depn't Info    Excess Blks
WILL RECOVER 31 BLOCKS BY COMPACTING DISK
WILL RECOVER 20 BLOCKS BY SHORTENING PROGRAMS
 1 BLOCK CURRENTLY AVAILABLE FOR STORAGE
* ORDERING IN PROGRESS
     1      0
     2      4
     3     14
     4     64
     6    134
     9    135
    10    143
     5    165
     7    170
     8    180
    11    197
    13    266   PRECEEDED BY A GAP OF 27 BLOCKS
    12    326
    18    344   PRECEEDED BY A GAP OF 4 BLOCKS
    14    347
* 'RET' TO CONTINUE
WRITE NEW INDEX (Y)?Y
INDEX REORDERED
SHORTEN ALL PROGRAMS TO MINIMUM SIZE (S)?S
BLOCK 0: ENTRY  LABEL      ADDR    # BLKS    TYPE    Type Depn't Info    Excess Blks
             1  :INDEX    ,     0:      4:      3:      4:     32:     32;
             2  :DOS      ,     4:     10:      0:     32:     32:     32;
             3  :BASIC    ,    14:     50:      1:  10752:     32;
             4  :BRANDEST,     64:     70:      3:     32:     32:     32;
NAME:        5  :P.BASIC  ,   134:      1:      4:      1:     32:     32;
             6  :DECODE   ,   135:      5:      2:      5:     32:     32;
             7  :ADTABL   ,   143:     19:      2:     19:     32:     32;
             8  :#CODE    ,   165:      2:      2:      2:     32:     32;
             9  :ERRORTRP,    170:      4:      2:      4:     32:     32;
            10  :D.BASC-T,    180:     15:      2:     15:     32:     32;
            11  :LABLTABL,    197:     39:      2:     39:     32:     32;
            12  :DATALT   ,   266:     60:      3:     60:     32:     32;
            13  :FLOWCHT  ,   326:     14:      2:     14:     32:     32;
            14  :ADDA     ,   344:      3:      2:      3:     32:     32;
            15  :SNGLSTEP,    347:      2:      1:   6912:     32;
BLOCK 1: ENTRY  LABEL      ADDR    # BLKS    TYPE    Type Depn't Info    Excess Blks
* 'RET' TO CONTINUE
BLOCK 2: ENTRY  LABEL      ADDR    # BLKS    TYPE    Type Depn't Info    Excess Blks
BLOCK 3: ENTRY  LABEL      ADDR    # BLKS    TYPE    Type Depn't Info    Excess Blks
WILL RECOVER 51 BLOCKS BY COMPACTING DISK
 1 BLOCK CURRENTLY AVAILABLE FOR STORAGE
* ORDERING IN PROGRESS
     1      0
     2      4
     3     14
     4     64
     5    134
     6    135
     7    143   PRECEEDED BY A GAP OF 3 BLOCKS
     8    165   PRECEEDED BY A GAP OF 3 BLOCKS
     9    170   PRECEEDED BY A GAP OF 3 BLOCKS
    10    180   PRECEEDED BY A GAP OF 6 BLOCKS
    11    197   PRECEEDED BY A GAP OF 2 BLOCKS
    12    266   PRECEEDED BY A GAP OF 30 BLOCKS
    13    326
    14    344   PRECEEDED BY A GAP OF 4 BLOCKS
    15    347
* 'RET' TO CONTINUE
INDEX CLOSE PACKED & IN ORDER
READY
```

65K RAM Memory Modification For The Sol-20

by Jim Spann

Don't give your Sol to the trash man; a savior is here! This simple modification gives the Sol Terminal Computer* a new lease on life by moving the VDM* and Solos* memory to the Sol's internal data bus (where it belonged anyway), so it no longer interferes with the S-100 memory address space. Now you can run all those big memory programs without having to buy a new computer, and still have access to all the Solos utility routines. And all old programs will execute properly.

A minor wiring change and the addition of two integrated circuits is required to extend the usability of the Sol Terminal Computer. This feat is accomplished by the use of a memory management flip-flop, controlled by a unused output port in the Sol. The parts required are a SN74LS74 (flip-flop) and SN7406 (open collector inverter).

The required circuit changes can be made without cutting any printed circuit board traces. The technique of hanging the IC pin to be changed outside of the socket and soldering a wire to it can save much wear and tear on the circuit board (Figure 1). The two chip memory management control circuit can be assembled on a small vector board and mounted inside the Sol under the keyboard as shown in Figure 2.

The following steps refer to the Sol schematics and drawings in the Sol manual. *Be sure to unplug and remove any S-100 boards during these steps.*

•Step 1. Build the custom memory management control circuit as shown in figure 3 on a small vector board. Set this board to one side. It will be used in a later step.

•Step 2. (This step moves the display memory data output signals from the S-100 bus to the Sol's internal bus.) Lift all the output pins of the tri-state I.C.s (see drawing 4) U29 and U89 and tie to the internal bus signal INT 0—INT 7, (see drawing 1). The internal bus runs all over the Sol mother board; use any handy INT 0—INT 7 signals to connect to. Be sure to mark-up changes and keep a accurate set of prints of your computing system.

PIN 13 of U89 (74LS367) to "INT 0" PIN 10 of U79 (74LS253)

PIN 11 of U89 (74LS367) to "INT 1" PIN 6 of U79 (74LS253)

PIN 13 of U29 (74LS367) to "INT 2" PIN 10 of U65 (74LS253)

PIN 9 of U29 (74LS367) to "INT 3" PIN 6 of U65 (74LS253)

PIN 7 of U29 (74LS367) to "INT 4" PIN 10 of U78 (74LS253)

PIN 11 of U29 (74LS367) to "INT 5" PIN 6 of U78 (74LS253)

PIN 3 of U29 (74LS367) to "INT 6" PIN 10 of U66 (74LS253)

PIN 5 of U29 (74LS367) to "INT 7" PIN 6 of U66 (74LS253)

•Step 3. (This step modifies the control of the internal/external multiplexer (U66, U78, U65, U79—see drawing 1) to allow the data from the display to get to the processor.)

Lift PIN 2 of U44 (74LS00) and tie it to PIN 1 of U44.

•Step 4. (This step moves the MWRITE signal of the internal RAM, so that it may be controlled by the memory management circuit.)

Lift PIN 9 of U44 (74LS00) and tie to PIN 14 of U46 (8T30). See drawing 4.

Lift PIN 13 of U24 (74LS04) and tie to PIN 14 of U46 (8T30). See drawing 2.

•Step 5. At this point the Sol computer should operate normally. Plug it in and try some programs that use the display; TARGET is a good test program. If the system does not work there is a wiring error, so double check everything and try again.

•Step 6. In step 6 the connection of the memory management circuit board is installed. Mount the memory management board and connect the circuit to VCC (+5 Vdc) and ground. This power comes from the Sol mother board. Connect to the following signals to the Sol. See figure 2 and drawing 2.

Jim Spann, Box 2061, Livermore, CA 94550.

* trademark names of Processor Technology Inc.

74LS74 PIN 1 (reset) to (S-100 signal POC) PIN 12 of U77.
74LS74 PIN 3 (clock) to (OUTPUT FCH) PIN 11 of U35.
74LS74 PIN 2 (data) to (S-100 signal DO) PIN 2 of U80.
7406 PIN 8 to PIN 3 of U34.
7406 PIN 1 and PIN 13 to PIN 6 of U23.
7406 PIN 2 to (S-100 signal MWRITE) PIN 11 of U50.
7406 PIN 12 to (S-100 signal FRDY) PIN 1 of U49.

This completes the modification of the Sol.

*Step 7. The system should operate normally; retest as in step 5. If there are any problems check the memory management flip-flop to make sure the Power On Clear (POC) resets it to a low level at PIN 5.

Theory of Operation

When the computer is first turned on the memory control flip-flop is cleared via the Power-On-Clear signal (POC). This signal is also generated when a restart is performed (holding both the upper case and repeat keys down). The Sol will operate normally with the Solos/display RAM/ROM memory block enabled.

The memory control flip-flop controls accesses (reads/writes) to the C000—CFFF hex memory block. This block 4K Solos/display RAM/ROM or a 4K RAM (can be part of a larger memory plane) memory on the S-100 bus. In other words the memory control flip-flop switches in the internal Solos memory or the external S-100 memory.

Operation

Software control of the memory management flip-flop is accomplished via the output instruction OUT FC and bit 0. If bit 0 is set to a zero (0) then this is normal Sol operation. If bit 1 is set to a one this enables the memory on the S-100 bus.

The programming example illustrates how to have a full 65K RAM system and use the Solos utilities with CP/M . The cold boot switches off the internal memory and turns on all RAM external memory.

Software Rules

These rules should be kept in mind when using this system.

1) Do not switch to the internal memory (Solos) if the STACK is in the C000—CFFF address area. Save the stack first, or the program will not be able to find its way back.

2) Do not switch to the internal memory from insided the C000—CFFF address area.

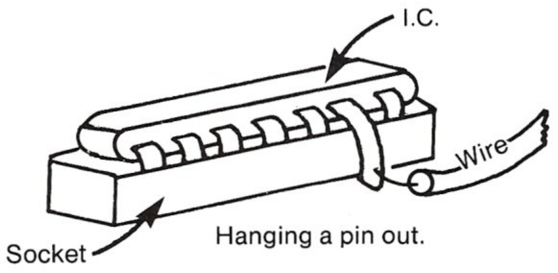


Figure 1.

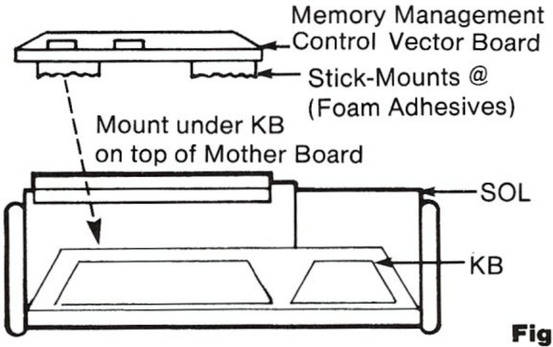

Memory Management Control Vector Board
Stick-Mounts @ (Foam Adhesives)
Mount under KB on top of Mother Board
SOL
KB

Figure 2.

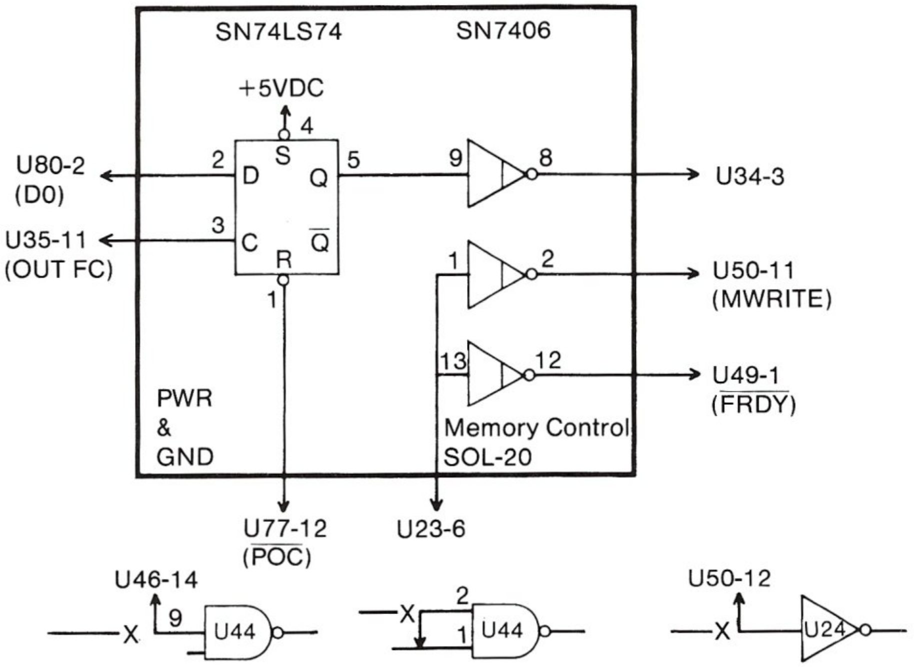


Figure 3.

The following is an example of a CP/M interface using the memory management modification.

```
;
memsw   equu    0FCH                ;output that control the memory
solon   equ     0                   ;solos on, normal operation
solof   equ     not solon and 0FFH       ;enables all ram
;
sinp    equ     0C01FH              ;SOLOS entry points
sout    equ     0C019H
aout    equ     0C01CH
ainp    equ     0C022H
stat    equ     0FAH                ;Keyboard status
char    equ     0FCH                ;Keyboard data in
;
boot:   lxi     sp,stack

        mvi     a,solof             ;turns off solos rom/ram area
        out     memsw
          .
          .
          .
        jmp     cpm;
;
wboot:  lxi     sp,stack

        mvi     a,solof             ;turns off solos
        out     memsw
          .
          .
          .
        jmp     gocpm
;
;    I/O ROUTINES
;=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
;
const:  in      stat                ;Sol KB
        cma
        ani     1
        rz
rtn:    mvi     a,0ffh
        ret
;
conin:  in      stat                ;Sol KB
        ani     1
        jnz     conin
        in      char
        ora     a
        jz      conin
        ret
;
reader: mov     a,1                 ;serial port
        lxi     h,ainp
        jmp     memctrl             ;send it to solos
```

```
;
punch:  mov     b,c                 ;for adout
        mvi     a,1                 ;serial port
        lxi     h,aout
        jmp     memctrl ;send it to solos

;       MEMCTRL
;       memory control
;       26 nov  79
; This program maps the solos area (C000 - CFFF) on & off.
; To allow for 64k byte operation of the Sol and still have
; access to the Solos software contain in the C000-CFFF area
; which includes the VDM, the following procedure is required.
;
; Power On Clear enables the Solos area so the first thing
; the boot program should do is turn off the Solos area.
;
; NOTE: THIS PROCEDURE MUST EXECUTE OUTSIDE OF THE C000-CFFF
;       ADDRESS SPACE.
;
; program calling sequence example:
;=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
conout  push    h                   ;save h we need it
        lxi     h,sout              ;vector to vdm in solos
        call    memctrl
        pop     h                   ;restore h
        ret
;=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
;
memsw   equ     0FCH                ;output that control the memory
solon   equ     0                   ;solos on, normal operation
solof   equ     not solon and 0FFH       ;enables all ram
;
memctrl shld    vector+1            ;store vector
        lxi     h,0
        dad     sp                  ;get stack
        shld    stkreg              ;save it
        lxi     sp,stack            ;get a local stack outside
                                    ;C000 - CFFF range
        mvi     a,solon             ;turn on solos so we have access
        out     memsw
vector  call    0                   ;get set on entry
        mvi     a,solof             ;turn off solos area
        out     memsw
        lhld    stkreg              ;recover org stack
        sphl
        ret
stkreg  dw      0
        ds      20
stack   dw      0

        end
```

Microsystems 1980-81 Article Index

This article index covers the preceding twelve issues of Microsystems. *Articles are listed by Volume-Issue-Page. Hence, "1-1-52" would be read as: Volume 1, Number 1, Page 52. Volume 1 covers the year 1980; Volume 2 covers the year 1981. The issues are as follows: No. 1 January/February, No. 2 March/April, No. 3 May/June, No. 4 July/August, No. 5 September/October, No. 6 November/December.*

Basic

Addressing The Cursor in Basic—Part I	L. Stein	1-2-34
Addressing The Cursor in Basic—Part II	L. Stein	1-3-30
Is Your Computer Out of Sorts	C. Terry	1-3-35
No More Waiting For Sorts	R. Sheffield	1-3-43
Modifications to CBasic-2	B&A Galewsky	1-6-28

Book Reviews

CP/M: A Self-Teaching Guide	S. Leibson	2-2-51
CP/M Handbook With MP/M	S. Leibson	2-2-52
CP/M Handbook With MP/M errors	A. Skjellum	2-2-16
CP/M Handbook With MP/M errors	A. Skjellum	2-3-18
The 8086 Book	C. Terry	2-4-62
CP/M User's Guide	C. Terry	2-6-66
Musical Applications of Microprocessors	J. Bondy	2-5-72

Communications

Computerized Bulletin Board Systems	R. Gorr	1-1-16
The Computer Communications Explosion	F. Lepow	2-3-20
Introduction To Data Communications	L. Huges	2-3-29
MODKOM Software Review	D. Thovson	2-3-34
COMMX and MCALL Reviews	G. Hart	2-3-36
S-100 Modems	M. Zeiger	2-3-40

Graphics

CGS-808 Color Graphics Board	J. Bondy	1-2-43
Dot Graphics On IMSAI-VIO	G. Sabot	2-2-30
Godbout Spectrum Graphics Board/Sublogic	J. Secondo	2-5-60
Modifying The SDS VDB-8024	J. Bondy	1-1-11
SSM VB3 Video Board	J. Bondy	2-2-26
Talos Digi-Kit-izer	J. Bondy	1-6-17

Hardware

Modifying The SDS VDB-8024	J. Bondy	1-1-11
Tarbell SD Disk Controller Mods	G. Lyons	1-2-46
A 16-Bit Wide Memory For S-100 Bus	K. Maum & P. Stakem	1-4-16
S-100 CPU Boards & Manufacturers	S. Libes	1-4-50
Running The 2MHz SD ExpandoRAM at 4MHz	W. Adams	1-5-44
Evaluating The New 8088 Microprocessor		1-5-52
S-100 Eprom Programmer Using 8255 PPI	T. Croal	1-6-46
The Other Processors For S-100 Systems	M. Cherlin	2-1-38
Tarbell SD Disk Controller Mods	G. Holz	2-1-53
MITS 88-S4K Memory Mods	W. Adams	2-2-48
Variable Speed Auto Slow Step For IMSAI	J. Long	2-4-58
An S-100 Clock/Calendar Circuit	F. Deadrick	2-4-64
The $150 48K S-100 Memory Board	R. Rodman	2-5-44

Hardware Reviews

CGS-808 Color Graphics Board	J. Bondy	1-2-43
ADS Noisemaker Board—Part I	S. Levine	1-5-50
Talos Digi-Kit-izer	J. Bondy	1-6-17
Double Density Disk Controllers	B. Weidmann	2-1-20
Casheab Music Synthesizer	J. Bondy	2-1-24
IDS 440 Printer	J. Bondy	2-1-30
Three New Generation Z80 CPU's Compared	B. Machrone	2-2-22
SSM VB3 Video Board	J. Bondy	2-2-26
Three S-100 Modem Cards Compared	M. Zeiger	2-3-40
TEC-86 8086 System	C. Terry	2-4-18
Seattle Computer Products 8086 System	B. Machrone	2-4-22
Alpha Micro 16-Bit System	H. Kee	2-4-26
Godbout Dual-Processor CPU Card	B. Ratoff	2-4-30
Televideo 920-C CRT Terminal	G. Hart	2-4-60
Godbout Spectrum Graphics Board	J. Secondo	2-5-60
Tarbell Double Density Disk Controller	F. Greeb	2-6-62

NorthStar System

Patches For NorthStar DOS 5.1	R. Reitz	1-2-10
A General Purpose Permuted Keyword Index	R. Reitz	1-3-18
Linear Programming (Pascal) Part I	W. Yarnall	1-2-14
Part II	W. Yarnall	1-3-24
Renaming Files on NorthStar Disk	M. Zeiger	1-5-38
A Spooling Program—Part I	R. Reitz	1-4-36
A Spooling Program—Part II	R. Reitz	1-5-34
Running NorthStar DOS & CP/M together	R. Reitz	1-6-32
Patching CP/M Disk on NorthStar System	T. Wiens	2-2-36
Input Queuing For NorthStar Double Density	R. Armstrong	2-4-54
NorthStar's New 5.2 DOS	S. Leibson	2-5-38
DOS/BIOS Directory & File Conversion—Part I	C. Young	2-5-48
—Part II	C. Young	2-6-40

Pascal

Eprom Burner Program (included in Modifying SDS-VDB-8024)	J. Bondy	1-1-11
Linear Programming Techniques in Pascal		
Part I	W. Yarnall	1-2-14
Part II	W. Yarnall	1-3-24
Pascal Speed Comparisons	F. Greeb	1-5-48
A General Purpose Permuted Keyword Index	R. Reitz	1-3-18
Monitor Program	J. Bondy	1-4-44
DOS/BIOS Directory & File Conversion for NorthStar systems		
Part I	C. Young	2-5-48
Part II	C. Young	2-6-40
Virtual Segment Procedures	J. Bondy	2-6-48
Managing Your Magazine File Program	L. Davis	2-5-65

Programming Techniques

Obscure & Dirty Programming Trick	B. Fuller	1-1-17
Linear Programming Techniques in Pascal I	W. Yarnall	1-2-14
Linear Programming Techniques in Pascal II	W. Yarnall	1-3-24

Addressing The Cursor in Basic—Part I	L Stein	1-2-34
Addressing The Cursor in Basic—Part II	L. Stein	2-3-30
Is Your Computer Out of Sorts (Basic)	C. Terry	1-3-35
No More Waiting For Sorts (Basic)	R. Shefield	1-3-43
Relocatable Code	R. Mossip	1-5-54
Ultra-Simple Index For Sequential Files	F. Gohlke	2-2-35
Selecting Functions Via Tables	F. Gohlke	2-2-42
New Life For ZTEL Text Editor	B. Machrone	2-2-44

S-100/IEEE-696 Standard

Importance of the IEEE-696/S-100 Standard	S. Libes	1-1-18
The IEEE-696/S-100 Standard		1-1-19
S-100 Bus—New Versus Old	S. Libes	1-2-39
S-100 Bus—Past, Present & Future—Part I	S. Libes	1-3-4
S-100 Bus—Past, Present & Future—Part II	S. Libes	1-4-4
S-100 Bus—A Progress Report	S. Libes	2-5-46

Software Reviews

Mate Text Editor/Word Processor	R. Graham	1-4-14
Information Master	B. Machrone	1-6-14
MODKOM	D. Thovson	2-3-34
COMMX	G. Hart	2-3-36
MCALL	G. Hart	2-3-36
CP/M-86	C. Terry	2-4-20
CP/M-86	B. Ratoff	2-4-30
16-Bit Disk Operating Systems	S. Libes	2-4-50
OS-1	D. Fiedler	2-5-26
NorthStar DOS 5.2	S. Leibson	2-5-38
BDS-C	D. Fiedler	2-5-30
BDS-C	D. Gewirtz	2-6-26
Small-C	D. Gewirtz	2-6-30
Tiny-C II	D. Gewirtz	2-6-32
Whitesmith's C	D. Gewirtz	2-6-38

CP/M

An Introduction To CP/M:		
Part 1: CP/M's Structure & Format	J. Epstein	1-1-6
Part 2: File Structure & Command Syntax	J. Epstein	1-2-28
Part 3: CCP Function	J. Epstein	1-3-12
Part 4: Utilities & BIOS	J. Epstein	1-5-10
The CP/M Connection:		
Part 1: Interfacing To Operating System & Relocating CP/M	C. Terry	1-4-32
Part 2: CP/M File Operations	C. Terry	1-5-24
Part 3: Implementing The IOBYTE Function	C. Terry	1-6-22
Part 4: Using CP/M Facilities in Your Own Programs	C. Terry	2-2-18
Part 5: A Real Application	C. Terry	2-3-46
A Directory Program For CP/M	M. Zeiger	1-6-38
Improved BIOS For Tarbell SD Controller	M. Nichols	1-2-49
Cold Boot Auto Program Load & Execute	L. Mohler	2-1-29
CP/M Bus—CP/M 2.2 Review	A. Skjellum	2-2-14
CP/M Bus—Features for CP/M—Part I	A. Skjellum	2-3-16
CP/M Bus—Features for CP/M—Part II	A. Skjellum	2-4-14
CP/M Bus—Features for CP/M—Part III	A. Skjellum	2-6-16
Choosing Between CRT & Printer Output	B. Kowitt	2-2-29
Patching CP/M Disk On NorthStar System	T. Wiens	2-2-36
CP/M-86 Review	C. Terry	2-4-20
CP/M-86 Review	B. Ratoff	2-4-30
BIOS For CP/M-86 (CompuPro Dual CPU & ICOM 3712 Disk Controller)	B. Ratoff	2-4-40

Software

An 8080 Disassembler	W. Yarnall	1-1-48
Linear Programming in UCSD Pascal—Part I	W. Yarnall	1-2-14
Linear Programming in UCSD Pascal—Part II	W. Yarnall	1-3-28
EPROM Burner Program (UCSD Pascal)	J. Bondy	1-1-11
Addressing The Cursor in Basic—Part I	L. Stein	1-2-34
Addressing The Cursor in Basic—Part II	L. Stein	1-3-30
Permuted Keyword Index Program (UCSD Pascal)	R. Reitz	1-3-18

Is Your Computer Out of Sorts (Basic)	C. Terry	1-3-35
No More Waiting For Sorts (Basic)	R. Sheffield	1-3-43
8080 Dynatrace (Assembler)	Foster/Meador	1-4-22
Spooling Program (Assembler) Part I	R. Reitz	1-4-36
Spooling Program (Assembler) Part II	R. Reitz	1-5-34
Monitor Program in Pascal	J. Bondy	1-4-44
CP/M BIOS Calls Demo Program	J. Epstein	1-5-18
Renaming Files on NorthStar Disk (Assembler)	M. Zeiger	1-5-38
Modifications to CBasic-2	B&A Galewsky	1-6-28
Directory Program For CP/M	M. Zeiger	1-6-38
A "Label-Basic" Preprocessor Program	G. Louis	2-1-42
Reading UCSD Pascal Disk Directories (Pascal)	J. Bondy	2-1-49
Dot Graphics On The IMSAI-VIO (Assembler)	G. Sabot	2-2-30
Patching CP/M Disk on NorthStar System	T. Wiens	2-2-36
A CP/M Print Utility	C. Terry	2-3-49
CP/M-86 BIOS	B. Ratoff	2-4-40
Input Queuing For NorthStar (Assembler)	R. Armstrong	2-4-54
Little Ada—Part I	R. Kenyon	2-5-14
Little Ada—Part II	R. Kenyon	2-6-54
The C Language—Part I	D. Gewirtz	2-6-20
A Disk Alignment Routine	L. Mohler	2-6-70

Software Directory

ACCESS/80	2-4-68
ACCOUNT 81	2-3-54
ALPHA FORTRAN	2-4-68
ALPHAL APL V2.0	2-2-53
APL	1-3-34
APL V2.3	1-6-54
APPAREL MANAGEMENT	1-6-54
BASIC-PACK	2-4-70
BDS-C	1-4-56
BEEF-UP	2-1-56
BILLING	2-1-56
CBS V1.1	2-1-58
COMM-4	1-4-54
COMMON	2-3-54
COMMX	2-3-58
COMPRESS	2-6-72
COMSTAR	2-2-53
CONST	2-6-72
D—DIRECTORY & STATUS	1-6-56
D80	2-2-53
DISCTIONARY	2-6-74
DF	2-2-56
DATA MERGE	2-3-56
DATABS	2-4-72
DOS/65	2-4-72
DIAGNOSTICS I	1-3-34
DISASMB	2-1-56
DISKTAPE	2-1-58
EDIT	2-3-56
ENCODE/DECODE	1-3-34
ENERGY BASIC	2-4-68
ENHANCED I/O DRIVERS	2-4-68
ENTRY	2-2-54
FORTH	2-2-54
FORTH-Timin Engineering	2-2-72
HAYES SYSTEM	2-2-56
HAM RADIO DX PACKAGE	1-4-56
HDBS	1-3-34
INFOMEDIA SYSTEM	2-2-54
INFORMATION MASTER	1-4-56
INVENTORY CONTROL	2-3-54
ISSCAI	1-6-54
LAYOUT	1-4-54
MASTER DISK CATALOG	2-5-74
MASTER LEDGER	1-5-60
MCALL	1-5-60
MDBS	1-3-34
MDBS-DRS	1-3-34
MENU	2-1-56
MFDT	2-1-58
MICRO LINK	2-5-74
MILESTONE	2-5-74
MULISP-79	1-4-55
MULTI-USER CP/M	1-5-58
MWP	1-5-58
MWP-SEL	1-6-54
ORDER-RIGHT	2-3-58
PLOTTER GRAPHICS	1-4-55
PRGM/MAP	2-3-54
PROGRAMMER'S APPRENTICE	2-6-74
PRO-TYPE WORD PROCESSOR	1-6-56
PROMER	2-3-58
PROPERTY ANALYSIS SYSTEM	2-2-54
SCREEN MASTER—Microcomputer Bus. Syst.	1-6-54
SCREEN MASTER—Marketing Essentials	2-6-72
SMALL-C	1-5-58
SORT 2.0	2-1-56
SPDES	2-1-56
SPEEDY DISK COPY	1-5-58
STAR*TRAC	2-4-70
TAPEDISK	2-1-58
TARBELL DD SUPPORT PACKAGE	1-6-56
TCS BUSINESS ACC'T PACKAGE	2-5-74
TED	1-5-58
UDE-PRT	2-3-58
UTILITIES (DMM-1)	2-6-76
VDRAW	1-4-54
VERS SORT	2-3-54
VIDEO ASM	1-4-56
VSELECT	2-1-58
WHATSIT?	1-4-54
WIREMASTER	2-2-53
WORD-C1	2-6-74
Z-80 DES	2-2-54
Z-80 FORTH	2-2-53
Z-80 DISK TEST	2-4-70
ZAS Z-8000	2-4-72
ZDM	2-3-54

Microsystems Reader Survey

Dear Readers:

Over the last two years, many of you have written to us. Your letters have offered encouragement (lots of it) and suggestions that have helped Microsystems *improve and grow. For that, we thank you.*

Our goal for 1982-83 is to bring you a bigger and better magazine. You can help us by filling out this questionaire and sending it back to us. We would like to know what types of articles you are interested in, what departments you have enjoyed, and which you feel could use improvement.

We would also like to know more about you—how you use computers, what kind of system you have, the role that computers play in your life, your level of expertise and some basic demographic facts. All this will help us produce a magazine that best meets your requirements as well as contact advertisers with products of interest to you. By the way, as we sell more pages of advertising, we are commited to balance them with more editorial material, and enlarge the magazine. You won't find Microsystems *running 75+% advertising.*

With your assistance we can bring you more of the kind of articles you find interesting, and keep Microsystems *on the right track. So please participate in our survey.*

Sincerely,

Claudette Moore

Claudette Moore
Managing Editor

Reading Habits

1 How did you initially become interested in *Microsystems*?
1 □ Talking with a friend or associate
2 □ Seeing a copy in a store
3 □ Receiving a promotional letter in the mail
4 □ Seeing an advertisement in another magazine

2 What is the average amount of time you spend reading or looking through each issue of *Microsystems*?
1 □ Less than 1 hour
2 □ 1 to 2 hours
3 □ 2 to 3 hours
4 □ 3 to 4 hours
5 □ 4 to 5 hours
6 □ over 5 hours

3 How many people read or look through your copy of *Microsystems*?
Myself and ______ others.

4 How long do you keep back issues of *Microsystems*?
1 □ Never discard them
2 □ Over one year
3 □ Six months to one year
4 □ Three to six months
5 □ Less than three months
6 □ Do not retain

5 Which computing periodicals do you regularly read or look through?
1 □ Microsystems
2 □ Kilobaud Microcomputing
3 □ Small Business Computers
4 □ Personal Computing
5 □ Creative Computing
6 □ Mini Micro Systems
7 □ Interface Age
8 □ Byte
9 □ Infoworld
10 □ Lifelines
11 □ Computer (IEEE)
12 □ Computer Design

Microsystems' Editorial Content

6 Rate the value of *Microsystems'* departments (on a scale of 1-5; 5 = most valuable, 1 = least valuable):
1 Letters: ______
2 News & Views: ______
3 Editor's Page: ______
4 CP/M Bus: ______
5 Software Directory: ______
6 New Products: ______
7 NorthStar Topics: ______

7 Rate your preference for types of articles:
Tutorials
1 CP/M: ______
2 Programming: ______
3 Hardware: ______
Software
4 Systems: ______
5 Applications: ______
Reviews
6 Software: ______
7 Hardware: ______
8 System: ______
Programs
9 Basic: ______
10 Assembler: ______
11 Pascal: ______
12 C: ______
13 PL/I: ______

8 What were the best four articles in 1981?
____
____
____
____

9 What other topics would you like to see covered in 1982?
____
____
____

Profession

10 What is your principal occupation?
1 □ Programmer/Analyst
2 □ Engineer/Scientist/Technician
3 □ Business Owner/Manager
4 □ Educator
5 □ Student
6 □ Professional (law, medicine, accounting, etc.)
7 □ Other

11 Do you use computers in your work?
□ yes □ no

12 If yes, with what frequency?
1 □ Daily
2 □ Weekly
3 □ Less frequent than weekly

13 If your company uses small comuters, please check the statement that best describes your relation to your company's purchases.
1 □ I decide what to buy
2 □ I influence the buying decision
3 □ I am not involved

14 What is your exact title?
____

Background

15 What is your age?
1 □ under 20
2 □ 20-24
3 □ 25-29
4 □ 30-34
5 □ 35-39
6 □ 40-44
7 □ 45-49
8 □ 50-54
9 □ 55-59
10 □ 60 and over

16 What is your sex?
□ Male □ Female

17 What is your personal income from all sources?
1 □ under $10,000
2 □ $10,000-$14,999
3 □ $15,000-$19,999
4 □ $20,000-$24,999
5 □ $25,000-$29,999
6 □ $30,000-$39,999
7 □ $40,000-$49,999
8 □ $50,000-$74,999
9 □ $75,000-$99,999
10 □ $100,000 or more

18 What level of education did you complete?
1 □ High school
2 □ Some college
3 □ Associates degree
4 □ 4 year college degree
5 □ Some graduate/professional
6 □ Master's degree
7 □ PhD
8 □ Professional degree (M.D., D.D.S., etc.)

19 Are you a member of a computer club?
□ Yes □ No

20 Are you a member of a scientific, engineering, or computer professional society?
□ Yes □ No

Microcomputing In Your Life

21 Which of the following best describes your involvement with personal computers? (check all that apply)
1 □ Personal or hobby
2 □ Professional or business
3 □ Educational purposes
4 □ Research and development
5 □ Other: ______________________________

22 Please check any of the following ways you purchase equipment:
1 □ Local computer store
2 □ Direct from manufacturer
3 □ Systems house or OEM
4 □ Mail order
5 □ Club group purchase
6 □ Other

23 What types of computer systems do you have access to? (check all that apply)

	Business Use	Personal Use
Large mainframe	□	□
Large timesharing system	□	□
Midi or large mini	□	□
Minicomputer	□	□
Microcomputer (S-100 bus)	□	□
Microcomputer (Apple, TRS-80, etc.)	□	□
Homebrew system	□	□

24 If you use an S-100 bus system, what manufacturers are included in your system?
(check all that apply)
1 □ Morrow Designs
2 □ Cromemco
3 □ NorthStar
4 □ Godbout Electronics
5 □ Vector Graphic
6 □ CCS
7 □ Tecmar
8 □ Imsai
9 □ Altair
10 □ Intel
11 □ ECT
12 □ Homebrew
13 □ SSM
14 □ Dynabyte

25 What is the approximate cost of your personal computer system(s)?
1 □ under $1000
2 □ $1000 to $2000
3 □ $2000 to $4000
4 □ $4000 to $6000
5 □ $6000 to $8000
6 □ $8000 to $10000
7 □ over $10000

26 Have you attended any computer fairs or shows in the past year?
□ Yes □ No

27 Please note the brands of products you intend to buy in the next 12 months.

CPU ______________________________

Printer ______________________________

Plotter ______________________________

Terminal ______________________________

Tape or Disk ______________________________

Memory ______________________________

Modem ______________________________

Graphics Pad ______________________________

Synthesizer ______________________________

Other peripheral ______________________________

Software ______________________________

28 How much do you plan to spend on hardware (computers, peripherals, etc.) and software within the next 12 months?
Hardware $______
Software $______

Microsystems' Ads

29 Do you find advertisements in *Microsystems* a valuable source of information?
□ Yes □ No

30 Which of these statements best describe how you approach advertisements in *Microsystems*?
1 □ Look for ads on specific products
2 □ Look at most ads
3 □ Generally look at half the ads
4 □ Rarely look at ads

31 Which of the following actions have you taken after reading an advertisement in *Microsystems*? (Check all that apply).
1 □ Asked for more information directly from an advertiser
2 □ Filed or made notes for future reference
3 □ Purchased or recommended the purchase of a product advertised
4 □ Purchased a competitive product

32 **Comments**

____
____
____
____
____
____
____

Please return completed form to *Microsystems* Survey,
39 East Hanover Avenue, Morris Plains, NJ 07950.
THANK YOU!

Prob-lem

The world is full of intriguing problems that never got into a textbook.

Problems for Computer Solution

by Steve Rogowski

Ninety intriguing and fascinating problems, each thoroughly discussed and referenced, make an excellent source of exercises in research and preliminary investigation. Eleven types of problems are provided in the following areas: arithmetic, algebra, geometry, trigonometry, number theory, probability, statistics, calculus and science. Author Stephen Rogowski of SUNY-Albany has included several problems which have never been solved. He feels that some research and an attempt to solve these will sharpen students' insight and awareness.

Some of the problems are not new like the one asking how much the $24 the Indians were paid for Manhattan would be worth today had it been deposited in a bank. However, this problem was revised to have a variable interest rate so it would be a challenge to program. Of course, many of the problems are new and have never been in print before.

The student edition has 106 pages and includes all 90 problems (with variations), 7 appendices and a complete bibliography. Cost is $4.95.

The 182-page teacher edition contains solutions to the problems, each with a complete listing in Basic, sample runs, and in-depth analyses explaining the algorithms and theory involved. Cost is $9.95.

To get one or both books send payment plus $2.00 shipping and handling per order to *Creative Computing*. Credit card orders may be called in toll-free to the number below.

Order yours today. If you are not completely satisfied, return it for a full refund plus your return postage.

creative computing

Morris Plains, NJ 07950
Toll-free **800-631-8112**
(In NJ 201-540-0445)

CP/M SYSTEMS
COMPATABLE 8080/Z80 SOFTWARE

STANDARD UTILITIES

		$	Source/Object
COMMX T.M.	— Menu Driven Communications with Information Services (DEC - IBM - UNIVAC - CBBS - etc.) and Remote COMMX Systems Transfer Any File Type/Size. Nine Link and Eight Local Functions Prompt for Mindless Operation.	250.00	75.00
D	— Disk Directory 4 Column Sort with File Size/Disk File and Space Status.	40.00	20.00
DDB	— Disk Directory Database UPDATE/INQUIRY Catalogs Files Fast.	60.00	25.00
DCOMP	— Disk File Compare with Another Disk File with Display Option.	30.00	20.00
MCOMP	— Memory Range Compare to Memory (ROM or RAM) - Console Logs Errors.	30.00	20.00
MTEST	— Memory Test Any Range with Before/After Write Error Bits + Pass #	30.00	20.00

ADVANCED UTILITIES

CDIR	— Comprehensive Sorted Disk Directory/Cross File Block Allocation Check.	30.00	20.00
COPSEQ	— Specify Disk Area and Copy Sequentially to CP/M File.	30.00	20.00
DASM	— 8080 Object Dis-Assembler with Symbol Table/XREF/ASCII MAP.	100.00	40.00
DXRSIZ	— Disk Exerciser Read or Write/Track/Sector/All/Set and Check Skew.	60.00	25.00
GEDIT	— Gang String Substitution Made Globally in One Pass Editor.	50.00	20.00
PREDIT	— Source Program Version Number Maintenance at Pre-Edit Time.	40.00	20.00
PROMER	— Load/Display/Patch/Copy/Verify/Burn ½K+1K+2K+4K Proms.	60.00	30.00
RELOC	— 8080 Object Code Relocator: Put This Into Your Program.	30.00	20.00
X6502	— 6502 Crossassembler MAC Macro Library and Post Processor.	100.00	

MEMORY MAPPED VIDEO

CGEN	— EPROM Character Generator Editor for Video Display Boards.	50.00	20.00
DXAM	— Disk Track Sector Examine with Update in HEX or ASCII or EBDIC.	40.00	20.00
VBASIC	— 9K Disk Basic with Super Video Commands and Full Screen Program Editor. Supports Different Video Cards with Identical Program Execution!	CALL	100.00
VGAMES	— For VBASIC: Othello/Blackjack/Breakout/Blockade/Poker Slot and Draw.	100.00	50.00
SOUNDS	— VBASIC Development System for AY-3-8910 Sound Chip Sounds.	75.00	30.00
PMIS	— Program Management Information System (Critical Path Method).	200.00	90.00
DBMS	— VBASIC Data Base Management System/Define/Enter/Report.	150.00	75.00
VIDEO	— Parameter Controlled Multi-User or Scroll Window Video Driver.	50.00	
VDRAW	— Vector Line Draw and Plot Subroutine for Fast Graphics.	30.00	
CHESS	— Graphic Games: IMSAI VIO: VECTOR GRAPHIC Flashwriter 2: SSM VB3.	30.00	20.00
INVADERS	— Zap!/Sound Effects/Joystick or Buttons or Console/Kill or be Killed.	50.00	20.00
STARTREK	— Realtime Action/Sound Effects with Host of Commands and Missions.	40.00	20.00
TARGET	— Moving Aircraft Shooting Gallery with Speed Options/Sound Effects.	40.00	20.00

*CP/M is a Registered Trademark of Digital Research

Disk $7.50 Extra — Cal. Residents Add 6% Sales Tax
Send Your Disk! — S. Den 8" +MICROP+APPLE+NSTAR
Dial 213/348-7909 to Get Free Product Brochure

HAWKEYE GRAFIX T.M.

23914 MOBILE
CANOGA PARK
CA 91307 USA

NEW! Speech synthesis using the Votrax* SC-01 with the

ads SYNTHETALKER!

- **64 PHONEMES AND 4 INFLECTIONS PROVIDE AN UNLIMITED VOCABULARY THAT IS USER PROGRAMMABLE**
- **PROGRAMMABLE OSCILLATOR FOR PITCH CONTROL GIVES YOU MORE NATURAL SPEECH OR MULTIPLE VOICES**
- **EXTENDED DEVICE OPTION**
- **UP TO TWO WAIT STATES**
- **EASILY PROGRAMMED IN BASIC!**

Write or call for complete details

Ackerman Digital Systems, Inc.
110 No. York Rd., Suite 208
Elmhurst, IL 60126
(312) 530-8992

*Votrax is a trademark of Federal Screw Works

Using CP/M's Undocumented "Autoload" Feature

by Kelly Smith

Ever wished you could just drop a diskette into your drive, boot it, and have it immediately start executing a program for you...or the kids want to play some nifty game, but you're tired of having your diskettes trashed because your 14 year old boy does not understand the full implications of ERA *.*...? Well there is a little known feature of CP/M that's worth knowing—it has a *built-in autoload*! I don't know why Digital Research fails to document this for public consumption, since they include the information in their OEM distribution documentation. After all, how many of us are OEM distributors? Well, here are the details of "autoload," and how to implement it on your CP/M system.

First, take a "scratch" diskette and do a full disk copy of your CP/M system diskette, including the system tracks. If you don't have a disk copy program, SYSGEN the "scratch" diskette and PIP all the stuff to it. Next, put the new diskette (to be set-up for "autoload") in drive A:, "warm boot" it (Control-C) and let's assume that we want to "autoload" Microsoft's MBasic and have it execute STARTREK. Here's what you do:

```
A>ddt movcpm.com<cr> <--- We need to "patch" MOVCPM.COM
DDT VERS 2.2
NEXT  PC
2800 0100            <--- Write down '2800' someplace
-d0a00<cr>           <--- Dump starting at address 0A00 Hex
```

Here is the start of CP/M (Version 2.2, but this method will work for CP/M Version 1.4 just as well):

```
     .← CCP jump entry (normal entry)
     |
     |        .← CCP jump entry to bypass "autoload"
     |        |
     |        |        .← 128 characters allowed for command
     |        |        |
     |        |        |  .← Number of characters in filename
     |        |        |  |
     |        |        |  |  .← Start of filename to "autoload"
     |        |        |  |  |
0A00 C3 5C 03 C3 58 03 7F 00 20 20 20 20 20 20 20 20 .\..X...
0A10 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48         COPYRIGH
0A20 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49 T (C) 1979, DIGI
0A30 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 TAL RESEARCH  ..
0A40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0A50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0A60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0A70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

O.k., now that we see that much, let's patch MOVCPM.COM to do the "autoload" of MBASIC.COM and STAR TREK.BAS...

```
-s0a07<cr>        <--- Substitute, starting at address 0A07 Hex
0A07 00 0f<cr>    <--- 15 characters in filenames to "autoload"
0A08 20 4d<cr>    <--- ASCII 'M'
0A09 20 42<cr>    <--- ASCII 'B'
0A0A 20 41<cr>    <--- ASCII 'A'
0A0B 20 53<cr>    <--- ASCII 'S'
0A0C 20 49<cr>    <--- ASCII 'I'
0A0D 20 43<cr>    <--- ASCII 'C'
0A0E 20 <cr>      <--- We needed a 'SPACE', so leave as is...
0A0F 20 53<cr>    <--- ASCII 'S'
0A10 20 54<cr>    <--- ASCII 'T'
0A11 20 41<cr>    <--- ASCII 'A'
0A12 20 52<cr>    <--- ASCII 'R'
0A13 20 53<cr>    <--- ASCII 'T'
0A14 20 54<cr>    <--- ASCII 'R'
0A15 20 45<cr>    <--- ASCII 'E'
0A16 20 4B<cr>    <--- ASCII 'K'
0A17 20 .<cr>     <--- Quit substituting
-d0a00<cr>        <--- Let's "Dump" what we just entered...
                              .← 15 characters in MBASIC STARTREK
                              |
                              |  .← Start of MBASIC STARTREK
                              |  |
0A00 C3 5C 03 C3 58 03 7F 0F 4D 42 41 53 49 43 20 53 .\..X...MBASIC.S
0A10 54 41 52 54 52 45 4A 20 43 4F 50 59 52 49 47 48 TARTREK.COPYRIGH
0A20 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49 T (C) 1979, DIGI
0A30 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 TAL RESEARCH  ..
0A40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0A50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0A60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0A70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Keep in mind, that you are not limited to just filenames—you could just as well "autoload" and envoke submit files (including command strings), whatever, and can completely "overlay" Digital Research's copyright notice and use as many ASCII character entries as are required. Don't substitute passed address 0A7F Hex, however! Now exit DDT (Control-C cr or G0 cr), and look at the number that you wrote down...it's there on that piece of paper, next to the beer can. Hmmm...says 2800...o.k., multiply the 2 by 16. Now, since the 8 ends in two zeros, subtract 1 from 8, and add 32 and 7 together to get...39! Now follow along closely:

```
A>save 39 trekload.com<cr>
```

We just saved 39 pages (256 bytes/page) of the patched MOVCPM.COM, and for reference, renamed it to TREKLOAD.COM. Now SYSGEN your diskette (however you do that) with TREKLOAD instead of MOVCPM, and "cold boot" the diskette.

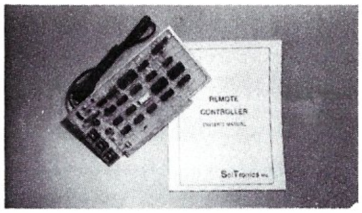
It should "boot" with MBasic being executed, then MBasic takes over and loads in STARTREK, and then you are ready to "zap" some Klingons! ■

Kelly Smith, 3055 Waco St., Simi Valley, CA 93063

Reprinted from CP/M NET NEWS.

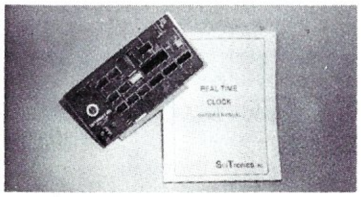
S-100 INNOVATORS:

SCITRONICS INC.



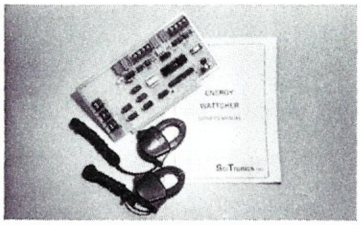
REMOTE CONTROLLER—Innovative Features:

- *Complete 256 address control—not just 16
- *No ultrasonic link—prevents erractic operation
- *120,208,240 and 277VAC control—for single & 3 phase operation
- *Hardware driven—requires minimal software
- *Complete line of industrial switches available—to 5.5KW



REAL TIME CLOCK—Innovative Featuers:

- *First to use LSI OKI clock chip
- *Crystal controlled for .002% accuracy
- *4 software selectable clock generated interrupts
- *Full clock and calendar data
- *Lithium battery backup good for 6000 hours!

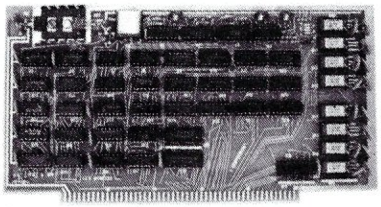


ENERGY WATTCHER™—Innovative Features:

- *First microcomputer based energy monitor
- *Clip on probes for easy installation
- *Monitors **Real Power**, not volt-amps
- *Peak Power and continuous power readings
- *Single and 3 phase operation

See your local computer dealer or contact SciTronics directly for more information. Watch for future innovative products from **SciTronics Inc., 523 So. Clewell St., P.O. Box 5344, Bethlehem, PA 18015 (215) 868-7220**

Super Compuprism Color Graphics



For the S-100 Bus. 32K of on board memory allows a 288 H. x 192V. dot matrix, for a total of 55,296 pixels. Every pixel is programable in any one of 16 colors or 16 grey levels completely independent of all other pixels in the matrix.

Compuprism Bare Board with documentation $45, kit $240, ass. and tested $280.
(16K Memory 144H. x 192V.)

Super Compuprism Bare Board with documentation $50, kit $350, ass. and tested $395.
(32K Memory 288H. x 192V.)

Add $15 to A & T price for 16 level grey scale.
Add $15 to A & T price for memory management port.

Compuprism software package, includes alpanumberics, point plot, line draw, and TRS-80* graphics simulation $20 or FREE with A & T unit.

Z-80 Users You Can Use TRS-80* Software

We offer an assembled hardware interface which we guarantee will load data from TRS-80* cassettes into any Z-80 based system. (Except sealed units.) The documentation explains how to patch the TRS-80* software to your system. In fact you can virtually change your Z-80 machine into a TRS-80* without making a single hardware change. The documentation also includes an example of patching SARGON II** into a Z-80 system.

The price is $30 or FREE with the purchase of an assembled compuprism or super compuprism unit.

A-D, D-A Board

S-100 board provides 16 channels of analog to digital input and 8 channels of digital to analog output. With on board kluge area. Total cost of board and parts less than $120. Bare board with documentation $45.

ALL COD ORDERS SHIPPED WITHIN 72 HOURS. 4MHz MOD FOR S.D. SYSTEMS. EXPANDORAM $10.

J.E.S. GRAPHICS, P.O. Box 2752
Tulsa, OK 74101, (918) 742-7104

TRS-80* is a trademark of Tandy Corp.
SARGON II** is a trademark of Hayden Book Co.

✓180

PASCAL

Announcing JRT PASCAL Version 2.
As the name implies, it's PASCAL refined — the second generation. Here's why:

- **No limit on program size — separately compiled external procedures are auto-loaded**
- **No limits of size of procedures, nesting levels or recursion**
- **14 digit precision arithmetic, no conversion errors, exponents from −64 to +63**
- **Dynamic strings up to 64K bytes**
- **Verbal error messages, dynamic trace, assembly code**
- **True dynamic storage with auto-compression**
- **All new 110 page user manual**
- **One-step compiler for CP/M* systems**

If you're considering any PASCAL, seriously consider JRT PASCAL Version 2 before you buy. Another advanced programming product from JRT Systems, it's ready for you now.

$295 Order from:
JRT SYSTEMS

Dept. 3c
P.O. Box 22365, 1891-23rd Avenue
San Francisco, CA 94122

*CP/M is a trademark of Digital Research

SOFTWARE DIRECTORY

Program Name: BPSXREF
Hardware System: CP/M with Microsoft Basic-80 v5.x
Minimum Memory Size: 48K
Language: Machine Code
Description: BPSXREF is a listing and cross-reference generator for Microsoft's Basic-80 5.x language. It produces a formatted program listing and alphabetized list of program variables and functions cross-referenced to the line numbers where they are used.

The formatted listing allows for page titles, page numbers and skipped lines for added clarity in program documentation. Options allow user to decide whether he wants a simple listing or only a detailed cross-reference, or some combination of listing and cross-reference.

BPSXREF operates on ASCII formatted CP/M files as produced by MBasic's SAVE command with the "A" option or text editors such as ED, WORDMASTER and MINCE. This is same file format required by Microsoft's Basic compiler, BASCOM.
Release: September 1981
Price: $124
Included with price: Disk and documentation.

82 Woods End Rd.
Fairfield, CT 06430
(203)254-1659

Program Name: SPELL
Hardware System: Standard CP/M and Heath/Zenith HDOS
Minimum Memory Size: 48K
Language: Machine Code
Description: SPELL is a spelling proofreader. It detects mispelled words in documents created by most text editors and word processors, including WordStar and Magic Wand. It allows listing unknown words, marking them in the document for easy editing, or adding them to the dictionary. Effective dictionary size is over 50,000 words with a user-expandable prefix/suffix table. SPELL processes 4,000 input words per minute.
Release: October 1981
Price: $49.95 plus $3 shipping/handling
Included with price: Disk and manual; specify 8" std CP/M or 5" Heath/Zenith CP/M or HDOS disk.
Where to purchase it:

The Software Toolworks
14478 Glorietta Dr.
Sherman Oaks, CA 91423
(213)986-4885

Program Name: Smartkey
Hardware System: Any CP/M system
Minimum Memory Size: 20K
Description: Smartkey installs a software interface between the console keyboard and CP/M, allowing the operator to 'redefine' key functions. Individual key codes may be altered and keys may be made to return a sequence of characters for each keystroke. The logical layout of keyboards may be improved and customized for particular applications software. Sets of key definitions can be saved on disk for re-use and definitions may be altered at any time. The program works with either version of CP/M and requires no hardware or software knowledge to install or use.
Release: October 1981
Price: $39.00
Included with price: 8" disk, 20 page manual.
Where to purchase it:

FBN Software
1111 Sawmill Gulch Road
Pebble Beach, CA 93953
(415)373-5303

Program Name: COMSTAR OVERLAY
Hardware System: North Star DOS
Minimum Memory Size: 32K
Language: Basic Compiler—Assembly language.
Description: An overlay structure is now possible under an extension to the COMSTAR compiler for North Star Basic. An overlay differs from page CHAINing in that root program segment and selected program variables can survive intact as a new program segment is introduced. An overlay structure allows very large programs to be executed and is also suitable for a menu driven system of programs. Includes a CP/M overlay capability for those with the COMSTAR-CP/M interface.
Release: September 1981
Price: $75.00 to registered owners of Comstar
Included with price: Modified Compiler, and overlay support routines.
Where to purchase it:

A.M. Ashley
395 Sierra Madre Villa
Pasadena, CA 91107
(213)793-5748

Program Name: ABSTAT
Hardware System: Any CP/M computer
Minimum Memory Size: 48K
Language: Pascal/MT+
Description: ABSTAT is an interactive statistics package. Commands include multiple linear regression, analysis of variance, cross tabulations, bar graphs, scatter plots, means tests and many others. Flexible data manipulation routines allow full data editing, subsetting, appending, and ASCII file transfer with straightforward algebraic equations. Up to twenty variables are accessible by name or number. There are facilities for

Microsystems — the CP/M* and S-100 User's Journal

CP/M is the software bus!+ S-100 is the hardware bus for sophisticated microcomputer users!

If you are a CP/M user, on any system—S-100, Apple, TRS-80, Heath, Ohio Scientific, Onyx, Durango, Intel MDS, Mostek MDX, etc—after all CP/M is the Disk Operating System that has been implemented on more computer systems than any other DOS—then *Microsystems* magazine is the "only" magazine published specifically for you!

Or, if you use an S-100/IEEE-696 based computer—and the most sophisticated microcomputer systems available use the S-100/IEEE-696 hardware bus—then *Microsystems* magazine is the "only" magazine published specifically for you!

We started publishing *Microsystems* almost two years ago to fill the void in the microcomputer field. There were magazines catering exclusively to the TRS-80, Apple, Pet, Heath, etc. system users. There were also broad based publications that cover the entire field but no one system in depth. But no magazine existed for CP/M users—nor did one exist for S-100 users.

The why and what of a software bus

First of all what is a "bus?" And why do we cali CP/M "the software bus?"

A "bus" is a technique used to interface many different modules. Examples are the "S-100/IEEE-696 Bus" and the "IEEE-488 Bus." These are hardware buses that permit a user to plug a bus-compatible device into the bus without having to make any other hardware modifications and expect the device to operate with little or no modification.

CP/M is a Disk Operating System (DOS). It was first introduced in 1974 and is now the oldest and most mature DOS for microcomputer systems. CP/M has now been implemented on over 250 different computer systems. It has been implemented on hard disk systems as well as floppy disk systems. It is supported by two user groups (CP/M-UG and SIG/M-UG) that have released over 80 volumes containing over 2,000 public domain programs that can be loaded and run on systems using the CP/M DOS. Add to this another 1,500 commercially available CP/M software packages and you have the largest applications software base in existence.

CP/M is the only DOS for micros that has stood the test of time (seven years) with the highest level of compatibility from version to version. And over the years this compatibility has been maintained as new features have been added.

This is why we say "CP/M is the software bus" and why *Microsystems* magazine is vital to providing CP/M users with technical information on using CP/M, interfacing to CP/M, new CP/M compatible products and for CP/M users to exchange ideas.

Why support the S-100 bus?

S-100 is currently the most widely used microcomputer hardware bus. It offers advantages not available with any other microcomputer system. Here are a few of the advantages:

S-100 is processor independent. There are already thirty different S-100 CPU cards that can be plugged into an S-100 bus computer. Nine 8-bit microprocessors are available: 6502, 6800, 6802, 6809, 2650, F8, 8080, 8085 and Z80. Eight 16-bit microprocessors are available: 8086, 8088, 9900, Z8000, 68000, Pascal Microengine, Alpha Micro (similar to LSI-11) and even the AMD2901 bit slice processor. Take your pick from the incredible offerings.

S-100 has the greatest microcomputer power. What other microcomputer system has direct addressing of up to 16 megabytes of memory, up to 65,536 I/O ports, up to 10 vectored interrupts, up to 16 masters on the bus (with priority) and up to 10 Mhz data transfer rate? You will have to go a long way to use up that computing power.

S-100 is standardized. The S-100 bus has been standardized by the IEEE (Institute of Electrical and Electronic Engineers) assuring the highest degree of compatibility among plug-in boards from different manufacturers. And, *Microsystems* has published the complete IEEE S-100/696 standard (all 26 pages).

S-100 has the greatest hardware support. There are now over sixty different manufacturers of about 400 different plug-in S-100 boards. Far greater than any other microcomputer system.

With all these advantages is it any wonder that S-100 systems are so popular with microcomputer users who want to do more than just play games?

F or the serious computer user.

Each issue of *Microsystems* brings you the latest in the CP/M and S-100 world. Articles on applications, tutorials, software development, product reviews, and lots more, to keep you on top of the ever changing microcomputer scene.

And if you are an S-100 system user using other operating systems (e.g. North Star) *Microsystems* also supports you.

Get your copy today

Order your subscription to *Microsystems*. Send $16 for one year (6 issues), $30 for two years (12 issues) or $42 for three years (18 issues). If you prefer, call our toll-free number, **800-631-8112** (in NJ 201-540-0445) to put your subscription on your MasterCard, Visa or American Express card. Canadian/Mexican and other foreign surface subscriptions are $19 and $35, respectively, per year and must be prepaid. We guarantee that you will be completely satisfied or we will refund your subscription.

Join thousands of *Microsystems* subscribers like Jim Johnstone of Los Altos CA, who said "*Microsystems* has lived up to your promises and my expectations. Congratulations."

the CP/M* and S-100 user's journal
MICROSYSTEMS™
a Creative Computing publication

39 East Hanover Avenue
Morris Plains, NJ 07950, USA
Toll-free 800-631-8112
(In NJ, 201-540-0445)

New subscribers are entitled to a **free** CP/M Reference Card. Be sure to ask for yours when you order.

* registered trademark Digital Research + registered trademark Lifeboat Associates

Software Shops

—Arizona—

Creative Software Systems: Systems integration and custom software (BASIC, PASCAL, Z-80 assembler). Small business and word processing systems. 632 Camelot Dr., Sierra Vista, AZ 85635.
Phone (602)458-6063.

—California—

Nelson Engineering: We write applications software for all micro-based systems in Asembly language, Basic, and Pascal. (213) 390-2963; 13450 Maxella Ave. G185 Suite 142, Marina Del Rey, CA 90291.

—Colorado—

Random Factors LTD.: Industrial test, control & data acquisition — Hi speed & accuracy for S100 & STD-BUS. From software to complete systems. W.K. Borsum, P.E., Random Factors LTD. Castle Rock, CO 80104. (303) 688-5338.

—Massachusetts—

MICROFT INC.: Customization of CP/M-80, MP/M, CP/M-86 and other operating systems. Full range of consulting services in microsystems software (systems, utilities applications), product selection, hardware. Contact: Tom Campbell, Chief of Technical Staff, P.O. Box 128, E. Falmouth, MA 02536. Phone (617)563-3807.

—New Jersey—

New Jersey Software Services: Full range of CP/M, S-100 services.
- System design
  - Business applications
  - Real-time systems
  - Mathematical analysis
- Software creation/customizing
  - 8080 assembly language
  - Z-80 assembly language
  - BASIC
  - FORTRAN
- Product evaluation/selection
  - Hardware Software
- In house training
- Telecommunication service
- Voice I/O applications

Contact C. A. Ryan, 6 Village Circle, Westfield, N.J. 07090 (201) 233-9297.

—New York—

Patrick Software Inc.: Systems design, professional advice, customization and programming services for CPM/8080, RSX/PDP-11 and others. 853 Carroll Street, Brooklyn, NY 11215. Phone: (212)622-8349.

Software Directory, continued...

writing formal reports and automatic batch processing from a command file. A help command is also provided.
Release: October 1981
Price: $400
Included with price: An 8" single density disk and 105 page manual.
Where to purchase it:
Anderson-Bell
2916 S. Stuart St.
Denver, Colorado 80236
(303)936-3859

Program Name: CATALOG
Hardware System: CP/M system with two 8" disk drives
Minimum Memory Size: 24K
Language: Machine Language
Description: CATALOG builds and maintains a compressed master data base containing information relevant to each file on each disk. Generating and updating this data base requires only information regarding what disk drive to read and what ID number to assign to the disk. CATALOG also permits users to enter short notes for each file and disk in data base. Data base query by filenames, filetypes, "wild cards," partial filenames or disk numbers as search directives.

The information displayed or printed by CATALOG shows the date they were last entered in the data base and the space used. File displays include filename, filetype, user number, system status, read-only status, file size, disk number containing that file and user-entered notes. A quick summary of all disks is also available which includes disk number, date last entered in the data base, space used and user-entered disk notes.
Release: October 1981
Price: $75 plus $2 shipping/handling, add tax in CA
Included with price: 8" Disk and Manual
Where to purchase it:
SRX Systems
2812 Westberry Drive
San Jose, CA 95132
(408)926-9411

Program Name: Tarbell Bios
Hardware System: 8080, Z80, 8085 computer, Double density controller
Minimum Memory Size: CP/M must be located 1K lower than memory size.
Language: 8080 source code
Description: Tarbell deblocked bios with virtual memory disk. Auto density select on single density, double density 51 by 128, and double density 16 by 512. Very fast! With Z80 running at 4MHz loads 25K in 2.5 seconds. The virtual memory disk is configured for banked memory boards using port 40h. The memory appears identical to a disk drive. Place a file in the memory disk and let Wordstar print it from the background. Disk waits disappear. Great for temporary files created from Pascal compilers, sort programs and etc.
Realease: September 1981
Price: $45.00
Included with price: COPY.ASM, FORMAT.ASM, BOOT.ASM and SYSGEND.COM Supplied on an eight inch single density disk.
Where to pruchase it:
Linmar
541 Ingraham Ave.
Calumet City, IL 60409
(312)868-4866 (Ask for Mark)

Program Name: RUNIC 1.0 Language Interpreter
Hardware System: 8" CP/M, TRS-80 Model II, H89 or Apple/CP/M
Mininum Memory Size: 48K
Language: Machine Code
Description: RUNIC has its roots in FORTH, but is much more approachable by the beginner and much more friendly to the user. Furthermore, RUNIC code is more easily read and maintained than FORTH code.

RUNIC implements higher level data structures than FORTH, including integers, floats, and character strings. RUNIC uses RPN to evaluate its expressions, but its control structures are much closer to those of Pascal, Basic, and other "algebraic" programming languages. READ, WRITE, and CLOSE words give RUNIC text file I/O, and a Tiny Filer (similar in concept to the UCSD Pascal Filer) allows file manipulation from the console. No source editor is supplied, however, source code may be prepared via ED, Wordstar, or any other CP/M text editor.
Release: October 1981
Price: $49.95 plus $3 postage/handling. NY residents add 7% tax.
Included with price: Disk and manual; speciify standard 8", TRS-80 Model II, H-89 SD or Apple-II CP/M disk.
Where to purchase it:
Starside Engineering
Box 8306
Rochester, NY 14618

Program Name: SMARTNET-DUMBNET
Hardware System: 8080, Z80 or 8085 running MP/M
Minimum Memory Size: 20K for satellites, 32K for the hub
Language: 8080 source code
Description: A network operating system that allows satellite computers to share common resources of a hub computer. The resources at the hub computer can consist of disk drives, printers, data bases, programs, etc. High performance operation is obtained because each user has a complete computer. DUMBNET is used with computers without disk drives and SMARTNET is used with computers with at least one disk drive and running CP/M 2.2. All functions of CP/M 2.2 are supported on the satellite computers.
Release: August 1980
Price: SMARTNET $150.00 DUMBNET $175.00; purchased together $300.00
Included with price: Complete documented source code and installation manual.
Where to purchase it:
LINMAR
541 Ingraham Ave.
Calumet City, IL 60409
(312)868-4866 ■

Get 12 issues of Creative Computing for the price of 8.

Some things are still cheaper by the dozen.

When you subscribe to *Creative Computing*, you get 12 issues for just $24.95. The same 12 issues would cost you $35 at the newsstand.

Why not enjoy *Creative Computing* all year long and save $10 at the same time.

To subscribe, call toll-free from 9 AM to 6 PM 800-631-8112. In New Jersey, call 201-540-0445. Or write to Creative Computing, P.O. Box 5214, Boulder, CO 80321. We accept Visa, MasterCard and American Express.

Creative Computing is the leading magazine of small computer applications and software. It has in-depth reviews of new systems, peripherals and software. Also articles for both beginners and experts; columns about popular computers, programming techniques and new products; and complete program listings for your computer.

Alvin Toffler says, "I read *Creative Computing* not only for information about how to make the most of my own equipment but to keep an eye on how the whole field is emerging."

Why not join over 160,000 subscribers and save money at the same time? Remember, they're cheaper by the dozen.

creative computing
the #1 magazine of computer applications and software
March
vol 7,
Emphasis on Education
Simulation of Competitive Business
MECC: An In-depth Report
Seymour Papert: Mindstorms
How to Select a Computer
How to Solve It
CAI Is Not Always Drill
Reading Level
Evaluation of
25 Fantastic New Products from the Consumer Electronics Show
creative computing software

NEW PRODUCTS

Display/S-100 Unit For Sorcerer Computers

A display/S-100 unit has been created to link the Sorcerer computer to all the manufacturers of S-100 bus products. The new unit combines an S-100 motherboard/ power supply within an attractive enclosure that also houses the video display (a 12" CRT with 20MHz bandwidth and green P31 phospor).

The Display/S-100 includes cables and documentation for easy installation. Suggested retail price is $699.00 F.O.B. Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, CA 94086; (408)734-9831.

Warp Drive Makes CP/M System Super-Fast

G & G Engineering has released WARP DRIVE, a hardware/software package which allows extended address S-100 RAM memory to emulate a disk drive under CP/M 2.2, providing speed increases of up to forty times over floppy disks. The WARP DRIVE system is based around the CompuPro dual processor 8085/8088 CPU board, allowing as much as 1 Megabyte of extended address RAM to act like a disk drive while running CP/M 80 on the 8MHz 8085, and to appear as directly addressable memory when running CP/M 86 on the 8MHz 8088. All CP/M 2.2 compatible software may be run unmodified on WARP DRIVE. G & G Engineering, 13708 Doolittle Drive, San Leandro, CA 94577; (415)895-0798.

6MHz CPU S-100 Card

The CP 600 Central Processor card uses a 6MHz Z80 and conforms to the IEEE-696 standard. Two on-board ports extend memory addressing to 24 bits and I/O addressing to 16 bits. This allows a possible 16 bytes of system memory and 65K of system I/O.

The System RAM refresh is done as a standard S-100 memory read cycle, minimizing the need for special logic on RAM cards. All eight lower address bits are used for refreshing to accommodate 64K dynamic RAM devices. A refresh localizer allows intensified parity checking in the area of currently executing programs. All bus cycles, including the refresh cycle, are three "T" times long. CP 600 has jumper-selectable memory and I/O wait states, as well as on-board EPROM wait. Ready signals are evaluated on the rising edge of PHI during BS per IEEE-696 standard. Echo Communications Corp., 1708 Stierlin Rd., Mountain View, CA 94043; (415)969-6086.

New Guide For Independent Software Authors

"Software Wanted: How and Where to Sell Your Program" is a new guide from Battery Lane Publications for anyone wishing to sell programs they've written themselves.

Many companies that market software are actively looking for programs written by independents. They know that they cannot possibly write all of the software packages that are needed by the fast-growing universe of micro and mini users, so they must turn to independent software authors. The new "Software Wanted" guide is intended to bring software authors and marketers together.

Throughout 1981, BLP has been surveying software publishers and distributors to determine what kinds of programs they buy, what royalties they pay, and which ones offer help with programming and documentation. That information, published periodically in the Computer Consultant journal, has now been collected into one volume. Information on over sixty companies is provided, along with tips on what to look for before signing a contract.

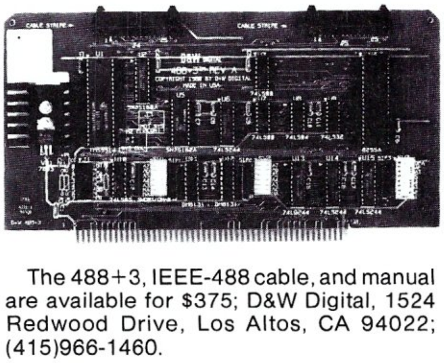
"Software Wanted" is being sold on a money-back guarantee basis, for $25, by: Battery Lane Publicatiions, PO Box 30214, Bethesda, MD 20814; (301)770-2726.

IEEE-488 To S-100 Interface

The 488+3 provides an IEEE-488 interface for S-100 (IEEE-696) computers. In addition, the 488+3 incorporates three parallel ports.

The IEEE-488 interface is implemented through the use of Texas Instruments' TMS 9914. The TMS 9914 communicates with the CPU via input/output-mapped 8-bit data ports. IEEE 488-1975/78 standard protocol is handled automatically in Talker, Listener, and Bus or System Controller operational modes.

The software I/O driver routines, written in MBasic, are supplied. These programs are callable subroutines for performing message handling and initialization. The manner in which they have been written allows them to be easily incorporated into a software program. No BIOS modifications are required.

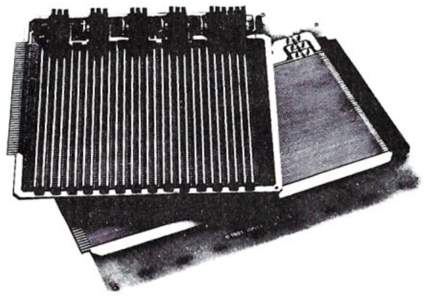


The 488+3, IEEE-488 cable, and manual are available for $375; D&W Digital, 1524 Redwood Drive, Los Altos, CA 94022; (415)966-1460.

S-100 Prototype Wirewrap & Extender Cards

Inner Access has introduced a 10" x 10" S-100 prototype wirewrap board (S100 PWWB-1) accomodating wirewrap sockets on .3" or .6" centers and a 13" x 10" Extender card. Both cards meet specifications for IEEE-696 compliance HH.

The 10" x 10" wirewrap board will accommodate 112 16-pin IC's plus a variety of 3M-style connectors. Onboard regulators provide 5 volts at 4 amps and ±12 volts at 1 amp.



The extender card has scope hangers on all signal leads and ground traces between all signals. Power traces have jumpers to allow current measurement. The edge connector fingers on both boards have gold-over-nickel plating for long life.

Wirewrap protoboard is $119 or $98 each for three, and the extender card is $58; Inner Access Corporation, 517-K Marine View, Belmont, CA 94002; (415)591-8295.

S-100 Typesetter Interface

The microCOMPOSER from Cybertext Corporation is an interface and software system which enables an S-100 system running CP/M to control CompuWriter typesetters (models I, II, IVa or Junior). CompuWriter and computer can function separately or together.

The word processing program in the microCOMPOSER system offers complete editing control: change letters, words, lines, paragraphs, pages; global control; format in galleys or pages. Page formats may include automatically inserted folios (starting at any number) and/or running heads or feet. Use computer printouts to proofread or revise before going to film. Save the copy on magnetic disks to reuse all or any part in other ways at any time. With the addition of one or more computer keyboards, multiple operators may generate copy on disks for printout on a single shared CompuWriter.

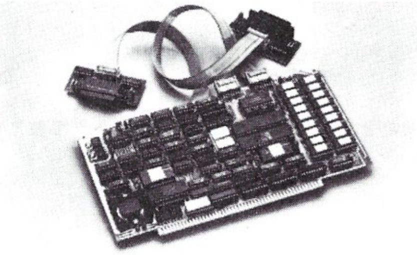
The system automatically prints the actual time required to print out copy, thus providing accurate figures for customer records and more definitive pricing information.

The microCOMPOSER system requires a simple "blackbox" connection inside the CompuWriter. There are only four easily located wires to attach (six in a Comp IV) and a plug connection to an existing connector. It is not necessary to make *any* alterations in the CompuWriter. A manual, describing in detail the installation plus all the operator information, is included. Contact: Cybertext Corporation, Box 860, Arcata, CA 95521; (707)822-7079.

S-100 Single Board Multi-User System

NET/82* is an S-100 board featuring a Z80A CPU, two serial ports, optional floating point processor, interrupt controller, shadow EPROM, real time clock, and an S-100 parallel port for communication with the master CPU. It includes bank-switched memory addressing and parity checking (both optional). NET/82 is compatible with MuDOS*, offered by MuSYS, as well as CP/M+, MP/M+ and CP/NET+.

Price: $1395; $1995 with 128K and floating point processor. NET/82 is available from MuSYS Corporation, 1451 Irvine Blvd., Suite 11, Tustin, CA 92680; (714)750-5693.



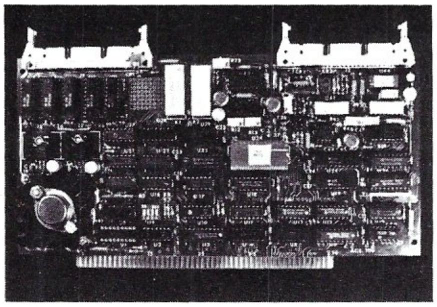
* Net/82 and MuDOS are trademarks of MuSYS Corporation.
+ CP/NET, MP/M and CP/M are trademarks of Digital Research.

New Products, continued...

Analog/Digital I/O S-100 Card

Automated Control Systems has introduced an S-100 card with both analog and digital I/O, the V.I.P.—Versatile Instrumentation Peripheral.

The analog portion is software-configurable and includes: an instrumentation amplifier input with resistor programmable gain, an analog gain block with trimmer adjustable offset, a 12-bit A/D converter offering 25 conversion time and software-selectable input ranges, a 12-bit multiplying D/A converter with simultaneous voltage and current source outputs, double buffered for signal synthesis capabilities.



The digital features include 8 TTL inputs and 48 TTL-compatible 30-volt, 100 mA open collector outputs. Selectable ROM decoding allows BCD programming of 5 digits of 7 segment LED displays. The board occupies 16 contiguous read and write I/O locations and offers switch-selectable addressing. There are full Z80 interrupt and WAIT state capabilities. Additionally, the V.I.P. has two uncommitted dual utility relays.

Price: $595 plus $4 shipping/handling, MA residents add 5% sales tax; Automated Control Systems, 1105 Broadway, Somerville, MA 02144; (617)628-5373.

Winchester/Floppy Controller For S-100

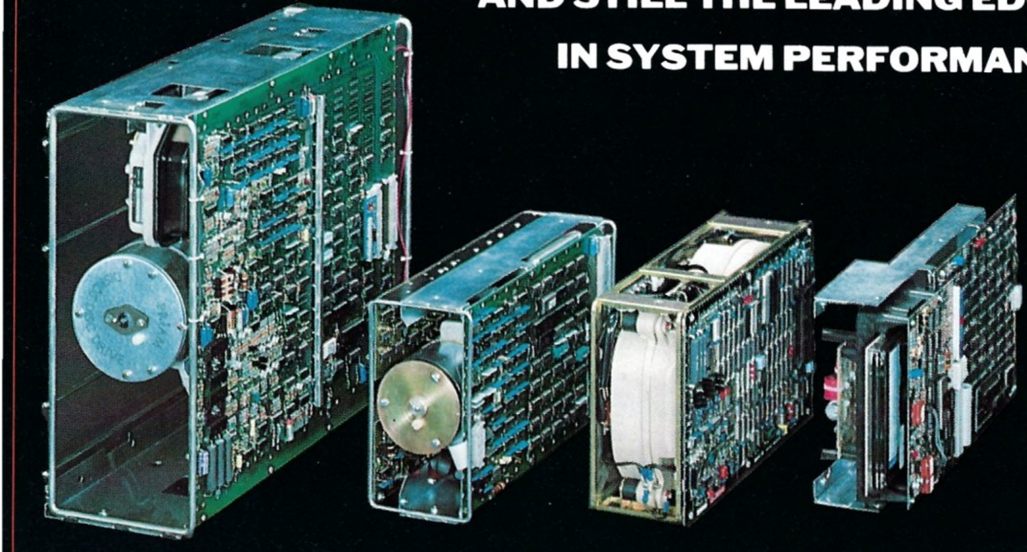
Many combinations of 5-1/4 or 8 inch Winchester and floppy disk drives can be controlled by the Piiceon D-100 single S-100 board disk controller, which is fully compatible with the IEEE-696/S-100 bus standard. It is I/O-mapped as an 8-bit port, assuring reliable data transfer with most S-100 systems. The D-100 operates with any drive using Shugart-compatible interfaces such as the Shugart SA-1000, SA-800, SA-400, the Seagate Technology ST-506, and the Quantum Q-2000. The D-100 is 100% compatible with IBM-formatted diskettes.

The D-100 responds to the command format of the NEC-765 floppy disk controller chip minimizing software rewrite for systems using NEC-765. For other systems, CP/M BIOS is available from Piiceon. It uses an 8 X 300 bipolar microcontroller chip. D-100 also provides an on-board sector buffer, a diagnostic LED, and CRC logic which support error checking on floppy disks and error correction on hard disks. Price is $745 from Piiceon Inc., 2350 Bering Dr., San Jose, CA 95131.

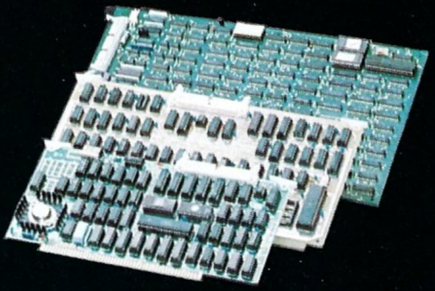
Advertiser Index

THE HARD EDGE

AND STILL THE LEADING EDGE . . .
IN SYSTEM PERFORMANCE



ADES



Hard Disk and Streaming Tape

reliability and versatility that can't be beat

OEM CONTROLLERS

- Multibus
- General Purpose
- S100

S33

• 31 or 62 MByte formatted

S10

• 10.5 or 32 MByte formatted

- Lowest cost/MByte from any manufacturer
- CP/M* compatible "drop in" BIOS
- Single S100 card controller
- Fully expandable up to four drives, 8" or 14"
- Reliable high performance Priam Winchester Disks
- Immediate delivery

SYSTEM 8

- 8" Winchester 10 or 32 MByte formatted
- Integrated streaming cartridge backup
- Streaming backup at 5 MB/min
- Selective file backup under CP/M
- Versatile parallel 1/0 or DMA interface

ADES

ADAPTIVE DATA & ENERGY SYSTEMS
2627 Pomona Boulevard • Pomona, CA 91768
Phone: (714) 594-5858

CP/M is a trademark of Digital Research Corp.