Programmer's Guide to CP/M

Edited by Sol Libes

Microsystems Press
A Division of Creative Computing Press
Morris Plains, New Jersey

Programmer's Guide to CP/M

Acknowledgements

I would like to gratefully acknowledge the assistance of the many authors whose articles appear in this volume. Without their efforts the book would have not been possible. Most particularly I would like to thank: Chris Terry, Bill Yarnall, Bruce Ratoff, Mark Zeiger, Charlie Foster, Bill Machrone, Randy Reitz, and Dave Fielder whose assistance extended beyond the articles which you see here. I would also like to express thanks to those who helped but whose names do not appear on the table of contents, namely: Russell Gorr and Fred Gohlke.

Sol Libes

Library of Congress Number: 82-61229
ISBN: 0-916688-37-2

Printed in the United States of America.

10 9 8 7 6 5 4 3 2

Creative Computing Press
39 E. Hanover Avenue
Morris Plains, NJ 07950

Preface

CP/M (Control Program for Microcomputers) is *the* most widely used microcomputer DOS (Disk Operating System) in use today. It is estimated that it is running on over 500,000 computers throughout the world. It has been implemented on more computers than any other DOS; from the IBM, Xerox, and DEC desktop computers to even Radio Shack and Apple personal computers and from systems as small as 16K of memory with no disks, to systems with several megabytes of memory and tens of thousands of megabytes of disk storage.

CP/M is supported by two user groups (CPMUG and SIG/M) which have released over one hundred volumes containing almost 3,000 public domain programs that can be easily loaded and run on systems using the CP/M DOS. Add 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 microcomputers that has stood the test of time. CP/M was created by Dr. Gary Kildall (PhD in Computer Science) in 1973 while he was working as a consultant to the Intel Corporation. Intel was designing a floppy disk interface to their 8080 microprocessor development system using one of the first 8" floppy disk drives Shugart had built. Prior to this, all storage of development software was done using a punched paper tape software storage system (the Intel Hex file format was developed specifically for this paper tape storage). The paper tape system, although very economical, was proving to be a very time consuming hinderance to software development. Gary was hired to create a high level software development language for Intel. Intel had others working on a DOS program. The result was that Gary wrote PL/M (Programming Language for Microprocessors), Intel's software development language. To test the performance of PL/M he used it to write a DOS which he naturally called "CP/M". Intel decided to adopt the DOS designed for them, leaving Gary to do what he wished with CP/M.

In 1976, after the introduction of several personal computers, Gary formed Digital Research Inc. and licensed IMSAI and Digital Microsystems, two small S-100 computer manufacturers, to distribute CP/M for their 8080-based systems with floppy disks. Needless to say these disk systems were an immediate success. Tarbell Electronics also offered a kit for an S-100 floppy disk controller card with which he included CP/M, at no extra charge. Thousands of computer hobbyists bought this card and interfaced it to systems such as the MITS Altair 8800, Processor Technology SOL and Polymorphics systems. Lifeboat Associates adapted CP/M for the North Star system and several others increasing the universality of CP/M. Lifeboat, with the help of computer hobbyists, also started the CP/M Users Group (CPMUG) which enabled microcomputer users to surmount the many problems involved in implementing CP/M and writing CP/M software. A second CP/M user group, Special Interest Group for Microcomputers (SIG/M) was formed in 1979.

This book is a collection of CP/M oriented articles which were originally published in MICROSYSTEMS magazine during the period January 1980 through February 1982 (Volume 1, No. 1 through Volume 3, No. 1). The articles provide an in depth look at CP/M from the programmer's viewpoint — namely the individual who is writing software which will interface directly to CP/M and the person who is installing CP/M on systems for which configurations do not already exist.

It is therefore not intended as a beginners book on CP/M. Individuals who wish to learn how to use CP/M may wish to consult the following books which are on a more introductory level:

J.N. Fernandez & R. Ashley, "Using CP/M, A Self-Teaching Guide", published by John Wiley & Sons, Inc., New York.

T. Hogan, "Osborne CP/M User Guide", published by Osborne/McGraw Hill, Berkeley, CA.

R. Zaks, "The CP/M Handbook with MP/M", published by Sybex, Inc., Berkeley, CA.

D.E. Cortesi, "Inside CP/M, A Guide for Users and Programmers", published by Holt, Rinehart and Winston, New York.

Contents

The CP/M Public Domain Software Libraries

Sol Libes

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.

Sol Libes is the editor of Microsystems magazine.

The SIG/M publishes an infrequent column which is carried on many of 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 SIG/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)746-4802 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)383-0470 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 Cribler 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
Hawaii		
Honolulu	96826	Victor Mori, (808)955-6683 2525 So. King St.
Illinois		
Glen Ellyn	60137	Stanley Hanson 187 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 Milton CBBS (617)864-3819 New England Computer Society P.O. Box 198
Littleton	01460	Nell Rosenberg 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 NCUG 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 Sullivan, (609)795-5607 Financial Software 54 Grove St
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)663-2642 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 Plafman, (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
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 Holtinwood 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 8551 16th Ave NW
Wisconsin		
Milwaukee	53213	Rick Martinek, CBBS (414)774-2883 Ricks Computer Center (414)774-8445 5903 West Parkhill

INTERNATIONAL

Australia		
Myrtle 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 68394
		Hans Stauffer c/o Eng. Eduardo Silassmis M. 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

Remote CP/M Software Exchange Systems

Sol Libes

In the previous section, I described the wealth of CP/M-based software that is available in the public domain via the CPMUG and SIG/M user groups, at very low cost. In fact most of this software, if not all of it, is in many instances available free of charge (if you do not count the cost of a phone call).

In addition to this software being available from the groups directly and from many local user groups (listed in the last issue) this software is available directly over the telephone line. There are a large number of computer systems operated as "remote" CP/M systems. They usually refer to themselves as either RCPM (Remote CP/M) or RBBS (Remote Bulletin Board System). These systems are operated mostly by individuals who donate their time, effort and their systems to the distribution of public domain CP/M software.

These systems operate primarily as bulletin board systems. Some cater to specific interests (e.g., C language, technical support, etc.). Some serve as a means for micro users in a local area to stay in touch.

In addition to their bulletin board functions, these systems all have facilities for uploading and downloading files. Many of these systems maintain several megabytes of files on-line always available to callers. To access these files and down-load them the caller just calls into the system (rarely is a password required) and follows the procedure that allows him to use the system as a standard CP/M system. A menu is usually given to guide the user.

Once the caller is into the CP/M system he can examine the directory of each disk on the system. To transfer a file, the user must use a transmission protocol that has become a standard on these systems. The protocol was created by Ward Christensen when he and Randy Suess created the first Bulletin Board system to go into operation. The protocol transmits files in 128 byte blocks, with a checksum at the end of the block.

The receiving system checks for errors, and if any are found sends a code back to the transmitting system to retransmit the block. This protocol is part of the MODEM program written by Ward and placed in the public domain via the CPMUG library. Subsequent versions, with enhancements, will also be found in the SIG/M library.

The RCPM/RBBS system has a program called XMODEM which the caller executes to put the system into the file transmit/receive mode of operation. Files can then be transferred between the two systems.

In addition to the RBBS and RCPM systems, file access facilities are available on the COMPUSERVE timesharing system. Although not free, it does provide another means for obtaining much public domain software. This system is part of COMPUSERVE's MICRONET service and is operated by three volunteers (see listing). It also includes a very active CP/M bulletin board. What is particularly interesting about the bulletin board system is that it includes technical representatives from MicroPro, Microsoft, Magnolia Microsystems, Tandy, and several other software and hardware suppliers. Users of the bulletin board can send messages directly to these companies and receive help directly. Not only that, one can read the messages going back and forth between these people..... most interesting! To access the CP/M bulletin board on MicroNet enter (at the command level prompt) "GO CIS-28" and then "R SIGS(CP-MIG)."

The following list is a highly condensed version of a list of RBBS systems I downloaded from the RBBS system in my local area.

Remote CP/M Software Exchange Systems

A list of Remote CP/M Software Exchange Systems using XMODEM for program transfers. Operators of new RCPM systems should send information about their systems to Ben Bronson [Hyde Park RCPM (312) 955-4493] or to Kelly Smith [CP/M-Net (805) 527-9321]. Revised October 25, 1981.

EXPLANATION OF CODES USED

BBS= Bulletin Board System
B1= 1 Baud rate available (300)
B2= 2 Baud rates available (110,300)
B3= 3 Baud rates available (110,300,1200)
B4= 4 Baud rates available (110,300,450,600)
B5= 5 Baud rates available (110,300,450,600,710)
CB= Call Back**
NALDS= No Alternate Long Distance Service*
RCPM= Remote CP/M system
S= Sprint*
M= MCI*
I= ITT*

NORTHEAST

Mississauga Ontario RCPM, (Toronto) 416-826-5394, Jud Newell. NALDS;B5,10Mb hard disk; 24 hrs; Sysop now has secondary system (with 2nd PMMI modem) integrated with main system so special arrangements can be made for extensive downloading. All vols of CPMUG and SIG/M software available on request. Interest in new & new versions of s'ware.

Mississauga Ontario HUG-RCPM, 416-271-3011, Toronto Heath UG. 1800-0900 wkdys, 24 hrs wknds; B5; NALDS;2+Mb files/5 drvs. Sysop plans 1Mb of HDOS s'ware & 1Mb CP/M software on line.

SuperBrain RCPM, 617-862-0781, Paul Kelly. 1900-0700 wkdys,24 hrs wknds; B3; S,I,M; 300K files on-line, [Lexington, Boston,MA area] (Interest in Superbrain-adapted CP/M programs)

Long Island NY RBBS, 516-698-8619, Tim Nicholes, CB, B3, 24 hrs; S,M; 1Mb files/2 drvs. Soon with 2 lines & modems, one half-duplex at 1200 baud)

Valley Stream NY RBBS/RCPM, 516-791-5041, Mike Schiller; 24hrs; B2; S,M, 300K files/2 drvs. May be running 212, 1200-baud modem.

Johnson City, NY SJBBS, 607-797-6416, Charles ---; Eves; etc. B1; 2Mb files/2 drvs. [Upstate New York]

Bearsville Town NY SJBBS, 914-679-6559, Hank Szyszka; B5; NALDS; 2MB files/4 drvs, [Upstate NY]. Installing MP/M. All CPMUG programs available by request.

Brewster NY RBBS, 914-279-5693, Paul Bosshold/Carl Erhorn;5pm-10pm, CB 10am-5pm, up 24hrs wknds; B4; NALDS; 500K files/1 drv. [Downstate New York] (S-100 based. General CP/M software)

Rochester RBBS, 716-223-1100, Arnie McGall; 24hrs; B2; S,M,I; 1.8Mb files/3 drls, (Upstate New York) S-100 based. General CP/M software. RBBS/RCPM system coexists with separate passwordad message system called Datastar, which can be entered from CP/M but runs on separate computer. 600 baud expected soon.

EAST CENTRAL

Cranford NJ RBBS/RCPM System, 201-272-1874, Bruce Ratoff, 24hrs, B4; S,N; 2-3Mb files/ 3 drvs. General CP/M software. Amateur Computer Group of NJ & SIG/M RBBS

Allentown Pa RBBS/RCPM System, 215-398-3937, Bill Earnest, 24hrs; B5; S,I.; 4.25Mb files/ hard disk(=4 logical disks). General CP/M software. Lehigh Valley Computer Club RBBS.

Baltimore Md Micro-Mail, 301-655-0393, Rod Hart; CB, Days/Eves until 2200; B5; S,I,M.; 1Mb files/2 drvs. General CP/M software; interest in Ham programs & modem s'ware in PASCAL & C)

Baltimore Md Prodigy Systems RBBS, 301-337-8825, NCB,24 hrs, B1, I,S,M. Down as of 10/16/81. 500K files/2 drvs.

Bel Air Md Nuclear RBBS/RCPM, 301-879-7841, Bob Loesch. NCB, 24 hrs; B2; N,I,S; 1.2Mb files/2 drvs. Down as of 09/21/81.

Grafton Va RBBS, 804-898-7493, Dave Holmes; 24 hrs; B1; NALDS; 200K files/2 drvs. (Tidewater Va.) CP/M, TRS-80 & Apple software; plans dual system (on one line) with LNW-80 & CP/M computer.

MIDWEST

Columbus Oh CBBS, 614-268-2227 (268-CBBS), Ben Miller; 24 hrs; B5; S,I,(M?), 300K files/3 drvs, running MP/M on a Tarbell SD controller; occasional slow response means Sysop also using system; interest in BDS-C programs.

Newark Oh RBBS/RCPM, 614-366-3252, Bo McCormick; 24 hrs; B1; NALDS; 500K files/2 drvs; Homebrew S-100. Interest in general software, offbeat graphics; other software may be requested from on-line master catalog.

Westland (Detroit) Mi RBBS/RCPM, 313-729-1905, Ron Fowler; CB; 24 hrsg; B4; S,M,I; 1.4Mb files/2 drvs. Emphasis very recent sfcwre.

Detroit Mi, Technical CBBS, 313-846-6127, Dave Hardy; 24hrs; B4; I,S,M; 3Mb files/3 drvs. Emphasis very recent releases. RCPM sysops desiring access to passworded RCPM Clearing House system should leave msg .

Royal Oak (Detroit) MI,CP/M, 313-759-6569, Keith Petersen; CB, 24hrs; B3; I,S,M; 600K on 2 drvs + 10Mb hard disk [=2 logical drvs]. Emphasis on new programs & recent updates of standard progs. 1200-baud 212-type modem available but not regularly on line, use CHAT or leave msg if you want PMMI switched out and 212 switched in.

MINICBBS/Sorcerer's Apprentice Group, 313-535-9186, Bob Hageman; CB; 24hr; B4; I,S,M; 500K/2 drvs. [Michigan] Running on an Exidy Sorcerer. Needs password, "SORCERER". Interest in adapting CP/M software and assorted hardware to Sorcerer systems.

Southfield MI RBBS/RCPM, 313-559-5326, Howard Booker; NCB, 24 hrs; 300/450 baud; I,S,M; 1Mb/2 drvs. Interest file directories/ catalogs of other RCPM systems & general s'ware)

Valparaiso In, Dick Hill's RBBS, 219-465-1056; 1900-2200 wkdys, 24hrs wknds; B4;NALDS; 4Mb files/4 drvs. S-100 based. General CP/M software.

Chicago Il, Calamity Cliffs Computer Center, 312-234-9251; 1400-0200 daily; B3; I,S,N; hard disk & 2 floppies. Many of CPMUG & SIG/M programs available by request.

Chicago Il, NEI RCPM System, 312-949-6189, Chuck Wiebeck; 1800-0100 wkdys, 1200-0100 wknds; B4; M,S,I; 2Mb files/2 drvs. Emphasis on communications programs, including versions adapted to non-standard CP/M systems.

Hyde Park Il [Chicago] RCPM/RBBS, 312-955-4493, Ben Bronson; 0100-1700 daily; B5; S,I,M; 2Mb files/2 drvs. Interest hard- & software reviews, C progs, and very recent releases of sid progs.

Chicago Il RAPM (Remote Apple CP/M), 312-384-4762, David Moritz; 24 hrs/7 days (sporadic); 300/450 baud; S,I,M; 250K files/2 drvs. Interest in telecom & other utilities for Apple/Softcard CP/M. 450 baud achieved using modified Hayes modem. Sysop may soon substitute a (TEI) S-100 system for the Apple.

Logan Square (Chicago Il) RCPM, 312-252-2136, Earl Bonkenfeld; 0100-1900 wkdys, irreg on wknds; B5; S,I,M; 1Mb files/ 2 drvs. Interest recent releases & developing on-line data-bases, with daily change of software on B drive.

Chicago Il HUG-CBBS, 312-671-4992, Paul Mayer & Dave Leonard; 2300-1900, 7 dys; B1; S,I,M; 2Mb files/ 2 drvs. H-89 based, operated for Heath-Zenith UG with interest in H19 & H89 adapted CP/M progs.

Palatine (Chicago Il) RCPM, 312-359-8080, Tim Cannon; CB; 1800-2400 wkdys, irreg on wknds; 300/1200 baud; S,M,I; 850K files/4 drvs. 212A 1200 baud modem.

Milwaukee Wi, Rick Martinek's System, 414-774-2681; Eves & wknds; B4; I,S,M; 1200K files/2 drvs.

SOUTH

Fort Mill SC, RIBBS, 803-547-6576, Bill Taylor; 24 hrs; 300/1200 baud; NALDS; 3Mb files/3 drvs. Heath/Zenith-based with 212 modem. Ham stuff, general s'ware, & on-line games.

Louisville Ky, RBBS/RCPM, 502-245-7811, Mike Jung; 0900-2100 wkdys, 24 hrs wends; B1; S,M; 2.5Mb files/5 drvs. Heath/Zenith-based. Emphasis on BASIC software & some HDOS stuff.

Huntsville Al, NACS/UAH RBBS/RCPM, 205-895-6749, Dan Wilkes; CB; 24 hrs; B4; NALDS; 700K files/4 drvs. No Ala Computer Soc @ U of Ala; general CP/M software.

CALIFORNIA

Bakersfield CA, CP/M-Net (tm), (805) 527-9321, Kelly Smith; 1900-2300 Mon-Fri, 1900 Fri-0700 Mon; B5; NALDS; 20Mb files/2 hard disks (=8 logical disks). System includes SIG/M Vol 1-10 =C:, SIG/M Vol 11-20 =P:, SIG/M Vol 21-25 = G:; XMODEM 'DISKMENU.DOC' for entire system directory (over 2100 files available).

Pasadena (Los Angeles area) CA, CBBS, 213-799-1632, Dick Mead; 24hrs; B5; I,S,M; 1.5Mb files/2 floppies & 8.3Mb hard disk.

Torrance (Los Angeles area) CA, RCPM, 213-549-9296, Dan Lopez/Alex Valdez; CB?; 1900-2300; B1; I,S,M; 500K files/2 drvs?. RBBS & other RCPM system progs available.

Palos Verdes CA, G.F.R.N.Data Exchange [RBBS], 213-541-2503, Skip Hansen; 24 hrs; 300/1200 baud; S,M,I; 2.4Mb files/2 drvs. Std CP/M s'ware. Interest in ham radio-related progs. Soon (with MP/M) will also be reachable thru 450 mhz radio.

San Diego CA,RCPM, 714-271-5615, Brian Kantor; 24 hrs; 300/1200 baud; T,S,M; 2.4Mb files/ 2 drvs; S-100 based with Auto-Cat modem. General CP/M s'ware with special interest in ham radio.

Siliconia (San Jose) CA, RBBS/RCPM, 408-287-5900, Paul Traina; M-Fdys 17:45-08:00, wknds 24hrs; B1; S,N,I; 2.4Mb files/3 drvs. S-100 (Godbout) based. Interest in PASCAL MT+ programs.

RBBS of Marin County (San Francisco area), 415-383-0473, Jim Ayers; Eves & nites wkdys, 24hrs wknd; B4; S,I,M; 1Mb files/2 drvs; S-100 (IMSAI) based; 24hr operation expected soon.

Larkspur (San Francisco area) CA, RBBS/RCPM, 415-461-7726, Jim C.; 24hrs; B5; S,I,M; 2+Mb/2 drvs. TRS-80/Omikron formerly used replaced by Godbout S-100 with PMMI.

Sacramento CA, CBBS/RCPM, 916-483-9718, Sacramento Microcomputer UG; 24hrs; B5; S; 700K+ files/2 drvs (expansion planned to 1.5M); Joe Bergin, Don Bozarth, John Moorhead, & Bud Ress Sysops. S-100 based; interest in CP/M; disks change bi-monthly.

NORTHWEST

Vancouver BC (Canada) CBBS, 604-777-7272, Steve Vinokouroff; 24hrs; B1; NALDS; 2Mb files/ 2 drvs. The system will be down for 6 months starting 10/15. When it comes up again next year it will have a new telno & PMMI modem.

Vancouver BC (Canada) Terry O'Brien's RCPM System, 604-584-2543

Olympia Wa. Yeln RBBS & CP/M, 206-458-3086, Dave Stanhope; CB; 24 hrs; NALDS; 500K files/2 drvs.

Portland Or, Chuck Forsberg's RCPM, 503-621-3193; 24hrs?; B3; NALDS?; 2Mb(?) files/2 drvs. Heath/Magnolia-based, 212a modem. Interest in C-language s'ware.

GENERAL NORTH AMERICA

CP-MIG on CompuServe MicroNet; type 'RSIGS (CP-MIG)';81; Sysops Dave Kozin, Tom Jorgenson & Charlie Strom are arranging to have MN carry much of new CPMUG and SIG/M software, plus newsletter and CP/M-oriented CBBS.

OVERSEAS

PERTH Western Australia Remote Computer/RBBS, Australian local; 09 457 6059, International; 619 457 6059. Trevor Marshall. Available most daylight hours & evenings. Manual connection only, requires CCITT 300 Baud modem in ANSWER mode for access. Running IOS (CP/M compatible), 64K Z80, 5Mhz system; 2Mb/2 drvs with 48K Cache buffer. All CPMUG and SIG/M volumes available by RBBS request. 1200 Baud Bell 202 will be available in 1 month.

Perth Western Australia, Paul Kelly's Remote Computer/ RBBS, Australian local; 09 459 3787; Available most evenings. Manual connection only; requires CCITT 300 Baud modem in ANSWER or ORIGINATE mode for access. Running IOS (CP/M compatible), 64K Z80, 5Mhz system; 2MB/2 drvs with 48K Cache buffer. All CPMUG and SIG/M volumes available by RBBS request 1200 Baud, Bell 202 will be available in 1 month.

---

* Alternative long-distance service should be considered when planning transfer of long programs. Charges on SPRINT, ITT/CITY-CALL and MCI are 50-60% of Bell's regular long-distance rates.

** Call-back systems; computer and real people share same telephone line. To contact people, dial & let phone ring until answered. To contact computer: dial, let phone ring once, hang up & re-dial.

All times listed are local time.

An Introduction to CP/M — Part 1

Jake Epstein

CP/M's Structure and Format

CP/M is an operating system written to run on 8080/8085/Z80 based microcomputers. It was written and is currently being supported by DIGITAL RESEARCH, PO BOX 579, PACIFIC GROVE, CALIFORNIA 93950. Their phone is 408-649-3896. Also there is a users group that provides a vehicle for the exchange of software, and currently, it has a quite large and varied list of languages and utility programs written to run under CP/M. There have been several versions of CP/M with the most current being ver. 2.0, and although at present I do not have this version, I will be getting it in the near future. The system that I have now is version 1.4 which DIGITAL RESEARCH continues to market and support. The prices that I have at this writing are $100 for ver. 1.4 and $150 for 2.0 which includes a FLOPPY DISKETTE and complete documentation.

In the past, CP/M has been limited to run mainly on 8 inch and 5 inch "floppies", for although there have been tape versions, the majority of use has been with disk-based systems. One of the abilities of ver. 2.0 is hard-disk compatibility which opens up the possibility of tremendous mass memory capability on a micro-computer. More on this in future articles. CP/M uses "SOFT-SECTOR" IBM 3740 format diskettes which store approximately 250,000 bytes of data in single density format on 8 inch disks and 70,000 bytes on the 5.25 inch variety, but double density is available on both systems and quad density is available for the 5.25 inch type. Thus, with a 4-drive dual-density system (DUAL REFERS TO THE ABILITY TO READ BOTH SINGLE AND DOUBLE DENSITY) the mass-memory size is 2 megabytes on 8 inch diskettes. Such large memory capability is simply incredible when one considers computer size and price tag.

What really makes this operating system so attractive for small system users, however, is that it is hardware independent in that once a user has it running, he/she can use CP/M software without having to interface it by writing drivers for his/her system configuration. This is made possible because a group of routines that comprise an area of the system called "BIOS" (BASIC INPUT OUTPUT SYSTEM) is implemented when the user first gets CP/M up and running, and then the operating system calls this area when it needs to do any I/O. Thus many combinations of terminal ports and disk controllers can be used. Currently, there are many disk controllers being manufactured, and several worthwhile articles could be written for this column by users of different boards. I have the TARBELL interface, and in a future column, I will discuss the board and my problems in getting it up and running. There are four other areas in the memory map of the system that I will mention here. BDOS (BASIC DISK OPERATING SYSTEM) has routines that are used by the system to access its system disk drives and thus provides for file management.

SEARCH	FIND A PARTICULAR FILE BY NAME
OPEN	PREPARE A FILE FOR OPERATIONS ON IT
CLOSE	CLOSE A PREVIOUSLY OPENED FILE
RENAME	GIVE A FILE A NEW NAME
READ	BRING IN A RECORD (1 SECTOR/128 BYTES) OF A FILE
WRITE	STORE ON THE DISK ONE RECORD IN AN OPENED FILE
SELECT	CHANGE THE LOGGED ON DISK DRIVE

DIGITAL RESEARCH also uses the term FDOS (FLOPPY-DISK OPERATING SYSTEM) which is actually the combined areas of BIOS and BDOS. The CCP (CONSOLE COMMAND PROCESSOR) can be compared to a keyboard monitor, for the user converses with the operating system via this area by using the command syntax of CP/M. In other words, the operator can accomplish various tasks such as list a directory of files on a disk, or execute a program file on a disk simply by using the terminal that is logged on the system. Finally, there is an area of memory that is used by programs that run under CP/M.

Jake Epstein, Box 571, Pittsfield, MA 01201.

Chapter 1
An Introduction To CP/M

An Introduction to CP/M — Part 2

Jake Epstein

File Structure and Command Syntax

In my last article, I discussed the basic memory map of the CP/M ver. 1.4 operating system plus the dialog that occurs when the system is initially "booted up". In this article I will be discussing the command syntax of CP/M, basic file structure, and FLOPPY DISKETTE mapping.

To begin, I will describe the layout of the FLOPPY DISKETTE so that terms and concepts that I use later will be clearer to those who are not yet familiar with this storage device. Also, for the sake of clarity and to prevent confusion, I will limit the discussion for now to 8 inch, single density diskettes. The diskette is a thin magnetic disk made up of material similar to that used in audio recording tape, and is housed in a square package that gives the disk both protection from dirt and support. When placed in a device known as a DRIVE, the disk rotates 360 times every minute and data is read from and written to the diskette via a read/record head that moves in and out depending on information supplied by host computer or device. What has made the floppy diskette so economically viable is that when actuated, the head comes in contact with the revolving magnetic material thus eliminating the mechanical difficulties associated with hard disks where heads have to be extremely close to the medium but cannot touch due to the injurious effects of abrasion. This is not to say that abrasion is not a problem for diskettes, for they can eventually wear out through normal use and dirt contamination.

Of special interest to CP/M users is diskette layout. The 8 inch variety contains 77 TRACKS which are actually concentric circular areas on the disk. When disks are initially formated, these tracks are laid out, thus read/record head alignment will prove very important for accuracy of data transfer. When a drive is commanded to read or write a certain track, the read/record head moves in or out (also known as SEEK) to find the specific track. In order to calibrate the drive, the head will perform a seek TRACK 0 upon system reset, and thus, all movement of the head can be monitored by software. The track at the outside edge of the diskette is track 0 whereas the innermost track is number 76. Each track is divided into 26 SECTORS thus the total number of sectors on the diskette is 77 * 26 or 2002. In order to locate specific sectors, the first sector of each track, sector 01, is indicated via a hardware indicator which senses a hole in the diskette, the INDEX. Other than TRACK 00, and the INDEX, the type of diskette used by CP/M, SOFT-SECTORED FORMAT, has no other hardware indicators of sector and track location, and thus alignment of the read head coupled with data encoded on the disk during FORMATTING aid in sector location.

There is a type of diskette that has a series of 32 holes to indicate sector location. These HARD SECTORED FORMAT diskettes are incompatible with CP/M and should be avoided even though they can be made useable through formatting tricks. Each sector of a properly formated diskette contains both areas for data (as stored and retrieved by the user) and areas for identification and error checking. It is beyond the scope of this article to discuss formatting, but investigation of the references appended to this text will provide information on this subject. For now, all one needs to know is that each sector contains the track number, sector number, an area for the storage of 128 data characters (BYTES) and a CRC (CYCLIC REDUNDANCY CHECK) location to aid in detecting errors. Thus the available storage on each disk is:

77 TRKS * 26 SECTRS/TRK * 128 BYTES/SECTR
= 256 256 BYTES.

For the uninitiated, a byte is a standard data size of the computer industry which is 8 bits long and represents 256 different BINARY or base 2 numbers. These numbers as stored on diskette can either represent numeric data, special codes such as machine instructions, or alphanumeric characters in the form of the 7 bit ASCII code. Because diskettes are organized as discrete data structures, the 2002 sectors, special characters and/or identification headings are not needed within the data to aid in its exchange. Thus in contrast to sequential storage systems such as MAGTAPE or PAPER TAPE, the disk operating system can handle large streams of data without needing to check for beginnings and/or endings. The disadvantages of disk systems is that the minimum number of data bytes that can be read or written at any one time is dependent on sector size. In contrast, serial systems, such as MAGTAPE, can read/write one character at a time even though this is rarely done. A result of disk storage is that there will be times when storage space will be wasted, for the amount of data stored may not use up an entire sector. An advantage is that having discrete structure allows for random access to files and/or parts of files. In other words, to access a file on a tape, one has to read the entire tape to find the desired data or start of data whereas in disk systems, data can be seen as a series of physical locations. Finally, the term used for the data stored in a sector (128 bytes) is RECORD; thus a record is 128 bytes in CP/M. Also, the term FILE is used to describe a set of data. In other words, the binary data that

This area, the TPA (TRANSIENT PROGRAM AREA), begins at memory location 0100H, and programs are written by the user to run so that they can access areas of BDOS, BIOS, and CCP to use routines already present in the system. Programs accomplish this through various "SYSTEM CALLS" and thus as I stressed above, a program that uses these calls can be run on any CP/M system. Finally, there is an area from 00H to 0FFH that provides for buffers and system memory registers (special storage areas) that are used by the operating system. Below is a memory map of a 16K CP/M system based on an INTEL MDS system BIOS as provided in DIGITAL RESEARCH'S documentation.

In the memory map, TBASE, CBASE and FBASE represent the memory address of the start of each area. TBASE is always equal to 0500 but CBASE and FBASE change with different sized systems. Systems can be made larger or smaller according to the amount of main memory or other considerations, but actually the memory map stays essentially the same with only the TPA changing in size for different versions. One final note on the operating system map is that a program may overlay the CCP or FDOS. What this means is that once the program is loaded from disk, it may use areas of memory previously occupied by the CCP or FDOS as long as they are not needed by the program.

On power up, the operating system is either entirely or partially loaded into memory from a CP/M system diskette via a bootstrap loader sequence. Any diskette that has the operating system on it and is in drive 0 is called the "SYSTEM DISKETTE", and if you have a multiple drive system, only the 0 drive needs to have the operating system on it. Tracks 0 and 1 contain the operating system while a directory of all the files on a disk and their locations and the data files are found on the other tracks (2-76 on 8" disk or 2-35 on 5.25" disk). Sector 1 track 00, the very first sector on the disk, contains a loader program, which when placed in memory from 00-7FH will bring in the rest of CP/M when executed. In order to read sector 1, track 0, a bootstrap loader is used that can either be toggled in via a "front panel" or burned into a ROM and executed. In my system which uses a ROM bootstrap, the entire process takes less than 5 seconds. In some systems, different from mine, the BIOS or parts of BIOS such as disk I/O is placed in ROM also. The BIOS contains boot programs to bring CP/M back in after a program has run, thus after executing a file, if BIOS is not altered, one need only execute the area of BIOS used to bring in the system. Location 000H in memory is reserved for a jump vector to this boot routine, thus to bring in CP/M, all one has to do is to jump to location 00H. If BIOS is altered however, the entire system has to be rebooted from the starting point using a computer reset command to get control via either a ROM monitor, or a CP/M bootstrap that is initiated on reset. The main advantage of jumping to location 00 and not reseting the computer is that the former procedure will not alter the TPA or in other words, the memory located from 100H to CBASE (start of CCP). Once back at command level, the user may then save a memory image of the TPA that may be examined, modified, or executed at a later time.

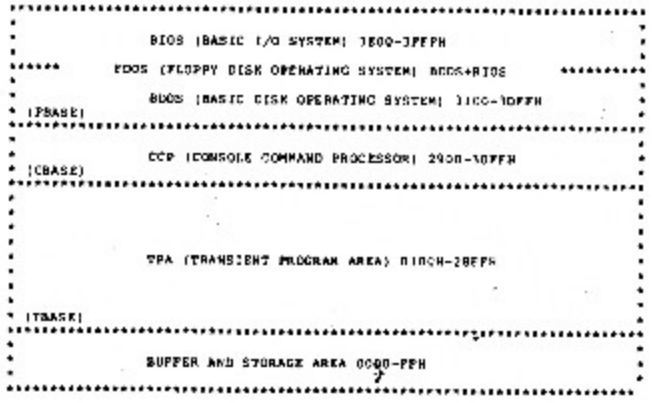
I have included the following literal flowchart to help summarize what happens when CP/M is brought in on a system after power up.

In my system which uses the "Tarbell" controller and BIOS, the following dialog occurs:

```
TARBELL 36K CPM V1.4 OF 11-13-78
2SIO VERSION.
HOW MANY DISKS? 2
A>
```

In the above, 36K is the CP/M memory size, V1.4 stands for version 1.4 of CP/M, and 2SIO VERSION is the type of I/O board used in my system. I typed in 2 to the question "HOW MANY DISKS?" because I currently have two disk drives, and finally the prompt "A>" signifies that drive "A" (drive 0) is the currently logged on disk. Also, memory locations 00-02 contain (C3 03 8A) which is the jump to the boot routine in BIOS as mentioned above, and locations 05-07 contain (C3 03 7D) which is the jump to BDOS to service system calls. All values are in hexadecimal.

Hopefully, the above text will give an idea of the structure and initial workings of CP/M on power up and reset.



1. TURN ON POWER		
2. PLACE SYSTEM DISKETTE IN DRIVE 0		
3. HIT RESET		INITIALIZES ROM BOOTSTRAP LOADER SETS DISK DRIVE 0 TO TRACK 00, SECTOR 1
4. HIT RUN		ROM BOOTSTRAP LOADER LOADS IN BOOT LOADER FROM SYSTEM DISKETTE TO LOCATION 00-7FH COMPUTER JUMPS OUT OF ROM BOOTSTRAP TO THE TO THE BOOT LOADER AT LOCATION 00 CCP, BDOS, AND BIOS ARE BROUGHT IN AND PLACED IN MEMORY AT THEIR RESPECTIVE LOCATIONS. THE COMPUTER JUMPS TO ROUTINE IN BIOS CALLED BOOT WHICH PRINTS A MESSAGE ASKS THE OPERATOR FOR THE NUMBER OF DISK DRIVES IN THE SYSTEM.
5. TYPE IN AT THE CONSOLE THE NUMBER OF DISK DRIVES IN SYSTEM		BIOS LOGS THE NUMBER OF DISKS, PLACE A JUMP INSTRUCTION TO A BOOT ROUTINE IN BIOS AT LOCATION 000, AND THEN PLACES A JUMP INSTRUCTION TO FBASE (START OF BDOS) AT LOCATION 005H WHICH IS FOR SYSTEM CALLS BY PROGRAMS IN THE TPA.
6. PROMPT IS PRINTED ON CONSOLE TERMINAL		SYSTEM IS READY FOR COMMAND PROCESSING

comprises a program such as a text-editor would be stored on diskette as a file, or, the text to this article could be stored as a file.

At this point I shall change the subject and discuss COMMAND SYNTAX. First of all, when the entire operating system is in main memory, communication is usually accomplished via a CONSOLE device such as a CRT terminal or video-keyboard interface. Through some hardware and software manipulation, however, other devices such as modem boards for communication over telephone lines or card readers for BATCH style processing can be used. The subsystem of CP/M that directs and processes this dialog is the CCP (CONSOLE COMMAND PROCESSOR), and it does so by forming an interface between the user implemented INPUT-OUTPUT routines found in BIOS (BASIC I/O SYSTEM), and file handling routines found in BDOS, (BASIC DISK OPERATING SYSTEM). In a future article, I will discuss BIOS handler modification for different hardware configurations, and user access of BDOS routines for generation of hardware independent programs that can be run on any CP/M or equivalent system.

Conversation when CP/M is in the COMMAND MODE (communicates via CCP with console device) occurs via uppercase alphanumeric characters and CONTROL functions. The lowercase alphabet is accepted, but it is converted to upper case before processing or storage, and thus user programs that leave lowercase characters in areas that are used by CCP, i.e. file names, can cause difficulties later. Also important to note is that all text is buffered until a carriage return function is received at which time a linefeed is sent to the console and then the command text is interpreted and executed. When a character is typed at the keyboard, it is read and then transmitted or ECHOED back to the screen or printer of the console. A BUFFER is an area of temporary storage that can be found anywhere information needs to be held, and the area of system memory in the CP/M memory map reserved for both console and disk file buffering is location 080H to 0FFH. Thus a text string of 128 characters can be entered before the buffer overflows causing an error to occur. When this happens, the entire text as typed in will be automatically sent to the console followed by a "?", and when the CCP does not understand a command due to a mistake in syntax or other error, the same type of echo of text occurs.

The following list of special characters are reserved and are only used in certain situations:

< > . , : = ? [ ]

The functions implemented via the CCP are almost all file handling in nature, whereas other functions such as memory modify or single step program execution are provided for in utility programs that run under, CP/M. Several of these are provided by DIGITAL RESEARCH on the original distribution diskette with the CP/M operating system, while others are available as separate software packages. The standard CP/M utilities will be discussed in future articles.

As discussed above, since floppy diskettes are organized as a series of discrete records, then special characters such as SOH (START OF HEADING) or FS (FILE SEPARATOR) and/or identification headings that are required to monitor sequential or continuous data streams are not required. However, in certain file types, CP/M utilizes procedures and conventions used in other systems. Below is a table of four type of files used by CP/M:

BINARY	Used for storage of MEMORY IMAGES of programs in machine code.
ASCII	Used for text or source programs. An EOF (END OF FILE) separator is used after last character. In CP/M this is a control-z (01AH)
HEX FORMAT	Byte values of 8 bits are converted to two hexadecimal values each of which represents one 4 bit nibble. Each nibble value is stored at an ASCII character.
RELOCATABLE	Used by certain assemblers or compilers. This is a special coding of machine programs that can be made to run at any memory location by using linking utility programs.

In the above file types binary is the most compact and can include any kind of data. What is meant by memory image is that the file is a copy of a block of data as it appeared in computer memory thus giving the user the ability to replicate a memory state at a future time after the computer has been shut off or the memory changed. The reason why an EOF is used with ASCII files is that this gives a method for retrieving an exact copy of a stored file as to length, for without this feature, the file would have to include all of the data that was unused from the last sector as explained above. Hex format is a very versatile data type in that software generated checks sums can be incorporated into the file giving a means for error checking and correction. This however, causes considerate software overhead and requires a great deal more storage space than other data types. Because Hex format can be confusing, below is an example of data as represented by different data types and hex format.

165 Decimal = 1010 0101 Binary = A5 Hex –

0100 0001 Binary = 41 Hex = ?? ASCII
0011 0101 Binary = 35 Hex = ?? ASCII

In the above the hexadecimal equivalent of the number, A5, is stored as ASCII codes so immediately it should be apparent that hex files will be at least twice as long as binary files. Checksums are generated in different ways, but usually all the data bytes in a BLOCK (a sub unit of a file usually associated with large storage devices like mag tape) are added together and then an adjusted number is stored in a non-data area of the file. When files are read and the stored checksum is different than the one generated, then an error has occurred. Some software systems have the means to correct errors. At present, CP/M can only detect errors by using hardware generated CRC (CYCLIC REDUNDANCY CHECKS) which are generated in a similar manner to checksums, but error correction is not available. Thus, using file types that use checksums can prove useful for increasing reliability of

disk storage. It should be mentioned that actually, CRC errors are detected by routines in BIOS, so that different disk controllers handle the errors differently.

CP/M uses a special file called the DIRECTORY to store pointers to file locations on the diskette. The directory files are located in 16 sectors on track 02. After a great deal of searching, I found that the following list of sectors contain the directory file on track 02:

sectors 1, 7, 13, 19, 25, 5, 11, 17, 23, 3
9, 15, 21, 2, 8, 14 (decimal)

The reason that the sectors are not in order, i.e. 1, 2, 3 etc., is that SKEWING is used to make reading and writing as efficient as possible. When two sectors are close together, hardware and software may not have time to identify and read/write the second one after doing so with the first before it slips by. In this situation, the system has to wait for one full revolution of the diskette for the second sector to come around again. The skewing for CP/M 1.4 is 6 but in other systems, it may be different causing incompatibility problems. In CP/M 2.0, this skewing can be modified by the user because this aspect of the system is handled in BIOS as opposed to version 1.4 where it is handled in BDOS. In a future article I will discuss this and other enhancements found in version 2.0. A word of warning to CP/M 1.4 users. If you have a CP/M system for 5.25 in disks and wish to add 8 inch disks, you will have problems because of sector skewing and track size. The same is true for 8 inch users that want to add the smaller drives. CP/M 2.0 eliminates this problem.

The data structure that stores information about each file on the disk is the FCB (FILE CONTROL BLOCK). Since I will be dedicating a lengthy discussion to the FCB in a future article on BDOS function calls, I will only describe a few concepts at this time. Each FCB has an area of 11 bytes in length that contains a PRIMARY and a SECONDARY name. The primary name can be any combination of up to 8 characters except for those that are reserved as mentioned above. Also, the primary name may be less than 8 characters, but when it is stored in the FCB, each empty position after the last character will be filled with ASCII "space" (20H). If a primary name is entered that is greater than 8 bytes, then it will be truncated upon storage. Names that are exactly the same as CCP function commands should not be used, for when files are accessed in the LOAD and EXECUTE function, the CCP will generate an error message ("?") because it will expect a command function. Below is a list of possible primary names:

PRIMARY NAME	REPRESENTATION IN FCB
TEXTEDIT	TEXTEDIT
BASIC	BASIC_ _ _
PASCAL785	PASCAL78
F-80	F-80_ _ _ _
1234	1234_ _ _ _

The following names are not allowed.

PRIMARY NAME	REASON
LETTER?D	? IS RESERVED
JACOB E	SPACE IN NAME
REN	REN IS A CPM FUNCTION

Secondary names are used to indicate certain types of files, and thus the CCP and/or utility programs can determine the data type of the file. For example, as I shall explain in the next articles, a file with the secondary name of COM can be LOADED into memory and then EXECUTED as a program. DIGITAL RESEARCH has reserved several secondary names for use in the operating system, but as software becomes more available and diverse, new reserved secondary names are added to the list. Of course, the user can use any secondary name that he/she desires, but if he/she uses a reserved name, then the file should fit the criteria of that file type. Below is a listing of the most used reserved secondary names:

NAME	DATA TYPE	USE
COM	BINARY	PROGRAMS THAT CAN BE EXECUTED
HEX	HEX FORMAT	OBJECT OF ASSEMBLERS OR COMPILERS
ASM	ASCII	SOURCE CODE FOR ASSEMBLERS
MAC	ASCII	SOURCE CODE FOR MACRO-ASSEMBLERS
BAS	ASCII	SOURCE CODE FOR BASIC COMPILERS
FOR	ASCII	SOURCE CODE FOR FORTRAN COMPILERS
PRN	ASCII	PRINTOUTS OF TEXT FORMATORS, COMPILERS, ASSEMBLERS, ETC.
SUB	ASCII	SOURCE FOR SUBMIT UTILITY
$$$	—	TEMPORARY FILE MAY BE ANY FORMAT
LIB	ASCII	LIBRARY FILES
TEX	ASCII	ASCII FOR TEXT-FORMATORS
DOC	ASCII	MESSAGES OR DOCU-MENTATION
MSG	ASCII	SAME AS DOC
TXT	ASCII	SAME AS DOC
REL	RELOCAT-ABLE	OBJECT OF RELOCATING ASSEMBLERS—SOURCE TO RELOCATING LINKERS

There are several that I left out, but in the following section, I will try to include as many as I know of. In naming files, the primary and secondary names are written together but separated by a ".". This delimiter is not found in the FCB but represents the position between the eighth and ninth characters in the name block. Here are a few examples. Please remember that "_" represents space or ASCII 20H:

FILE NAME	FCB REPRESENTATION
BASIC.COM	BASIC_ _ _COM
FORTRAN8.DOC	FORTRAN8DOC
LETTER.1	LETTER_ _ _1_ _

A binary dump of a FCB name block with ascii equivalents would be:

005D42 41 53 49 43 20 20 20 434F 4D BASIC COM

where 005D is a location in system memory where the file

name usually occurs and 42 41 etc. are the hexadecimal equivalents of ascii codes.

An area that can cause a great deal of confusion is ambiguous verses unambiguous file names. These terms refer to the ability of the CCP to work with files with similar but not identical names. Two special characters are used: "?" and "*" also known as "wild card". Ambiguous file names use these characters whereas unambiguous do not have them present. Also, file names as found in FCB's are unambiguous. Although I will get into more detail in next month's article when I discuss CCP command functions, the following example and explanation will hopefully give a better understanding of this concept.

ASM?.COM can describe —
ASM1.COM or
ASMZ.COM or
ASMA.COM

JANE.??? CANDESCRIBE —
JANE.COM or
JANE.123 or
JANE.TEX

The wild card "*" character is used to replace an entire primary name, secondary name or both.

** CAN DESCRIBE —
FORTRAN.COM or
LETTER.TEX or
TEST.DAT

These two characters are used mainly in listing out the names of files on a disk. For example, using the name *.COM with the DIR command of CCP would list all COM files.

The final aspect of CP/M file structure that I wish to discuss is sector allocation and file size. Each file control block has space for 16 data bytes. This list is referred to as the DISK ALLOCATION MAP in the CP/M documentation, and it is used by BDOS to find the ordered list of sectors comprising the file pointed to by the FCB. After a great deal of analysis, I discovered that each position in this table represents a block of 8 sectors. Block numbers 00 and 01 contain the directory as I listed above, but blocks 02 and up point to data files. BDOS also uses another byte in the FCB that keeps track of the total number of records (sectors) in each file in the event that not every sector in a block is used. For example, if a file needs only 3 sectors, then the other 5 in the block pointed to by the disk allocation map are unused. This phenomemon is the same as that discussed above in reference to unused sector space. By comparing actual file size with total available space, CP/M has a means of managing disk space for small file lengths. Since there are 16 positions in the disk allocation map, then the following figures can be calculated for the maximum storage capacity for one file pointed to by one FCB:

16 blks * 8 sectrs/blk * 128 bytes/sectr =
16 384 bytes

When a FCB becomes full, the byte containing the number of records becomes equal to 80H (128 decimal) and then BDOS creates (during write functions) or searches (during read functions) for a new FCB with the same file name as before. This new FCB is called an extension and BDOS is able to create or read up to 15 extensions. Thus in CP/M ver. 1.4, files can be created that are 16 * 16 384 or 262 144 bytes in length. Of course as calculated above, this is impossible because there are only 256 256 bytes of storage on a single density disk. The total amount of storage available for data is calculated below:

256 256 bytes/disk
- 6 656 bytes/tracks 0 and 1 (system tracks)
- 1 024 bytes/directors file

248 576 bytes

Since each block in a disk access table points to 8 sectors, then this total length in bytes is 1024 (128 x 8). When files are shorter than 1024 bytes or the last block of a file is not full, then this unused space will be wasted. If however, there were 64 files each a maximum of 16 384 bytes in length as calculated above, then total storage would appear to be 64 * 384 or 1048 576 bytes. (The reason why I chose the number 64 is that the total length of a FCB as found in the directory is 32 bytes, thus each sector can contain 128/32 or 4 FCBs. 16 sectors in the directory * 4 gives 64.) This is of course quite a bit more than the disk can store. Actually, the total storage space is determined by the fact that CP/M 1.4 supports 243 blocks, and since blocks 00 and 01 are used by the directory, the maximum storage on a disk when there is no unused space is 241 * 1024 or 246 784 bytes. Finally, the CCP command STAT will give the unused space on a disk. This is given in 1000 byte increments thus a disk with no files in the directory will appear to have 241K (K=1000) instead of 246K and the size of individual files will given to the nearest 1000 above the actual size. For example, STAT will give the size of a file of 256 bytes (2 sectors) as 1K, (the full block).

In conclusion, I have included a great deal of information that may or may not prove useful at this time to all users of CP/M, but hopefully, it will help you to expand your knowledge of file structure and management. In the next section, I will get into more practical matters.

References:

"CP/M 2.0 User's Guide", (Set of 7 Manuals), Digital Research, Pacific Grove, Ca.

Edelson, Roger: "Super Chip FD1771"," Interface Age, vol. 1, nos. 11, 12, vol. 2, no. 1., Oct-Dec 1976.

"Series 400 Floppy Disk Drive Operation & Maintenance Manual", Innootronics, Lincoln, Ma.

Horowitz, E. and Sahni, S.: "Fundamentals of Data Structures", Computer Science Press, Inc., 1977, 564 pp.

Wiederhold, Gio: "Database Design", McGraw Hill, Inc., 1977, 658 pp.

FD1771 A/B — 01: "Floppy Disk Formatter/Controller Data Sheet", Western Digital, Newport Beach, Ca., Mar 77.

An Introduction to CP/M — Part 3

Jake Epstein

CCP Functions

In this section, the third in a series on the CP/M operating system, I will be discussing the practical matter of console operation of CP/M. I have also included a section on mass-storage configurations available to CP/M users.

Once the CP/M operating system is 'booted up', the user has two options that can exercised. One is to execute the various commands inherent in the CCP, (CONSOLE COMMAND PROCESSOR). The other is to execute a program that has been stored as a file on the disk. While functioning in the CCP mode, the syntax of CP/M, as discussed in Article II, will prevail, but once a program is executed, then console syntax may change.

The 7 commands built into the CCP are shown in Table 1:

TABLE 1 - CCP COMMANDS

COMMAND	TYPE	FUNCTION
ERA	Alter	Erase a FCB in the directory
DIR	Non-alter	List files in the directory
REN	Alter	Rename a file
SAVE	Alter	Save memory image as a file
TYPE	Non-alter	Type contents of a file
(LOAD FILE-EXECUTE)	Non-alter	Load file in TPA then execute code at 100h
USER	Non-alter	Set user number, ver. 2.0 only

In the above list, functions that alter will change contents of a disk, and thus, care must used when exercizing commands that do so or data may be lost. Once data has been erased, it cannot be recovered so an important chore that users must do is make backup copies of files that are important in case of accident or mistake in command usage. More on this later.

Before explaining each built-in command, I will first describe disk log-in commands. As described in Article I, when the system is initially booted up, the prompt A> appears. This indicates that as far as the operating system is concerned, the storage device named A is online and ready to function as commanded by the user via the CCP. In the computer field, two terms are used to describe I/O devices: LOGICAL and PHYSICAL. Physical is a term referring to the device as it actually ocurrs in the real world. Logical refers to devices as they are seen by software. The following list should clarify the differences.

PHYSICAL	LOGICAL
8 inch floppy disk	A:
5.25 inch floppy disk	B:
1600 bpi mag-tape	C:
CRT	CON
ASR 33 teletype	LST
Paper tape	RDR

When there are several physical devices of the same type, then numbers are used beginning with 0. In other words, drive 0, drive 1, drive 2, and drive 3 would be the physical devices in a computer system with 4 floppy disk units. On the other hand, when the the user wants to access any of these via the operating system, then the logical device name is used. The value of this is that physical matters are taken care of by hardware/software interfaces found in the operating system leaving the user free to concentrate on other functions that use the logical devices.

In CP/M 1.4, BDOS (BASIC DISK OPERATING SYSTEM) and BIOS (BASIC INPUT/OUTPUT System) both contain software that is dependent of disk type, density, and size. As discussed last month, sector skew is a function determined in BDOS thus CP/M for 5.25 Inch disks will not function with 8 inch and vice-versa. Also, all disks in a system have to be compatible with the mixing of disk types impossible. A big advantage of CP/M 2.0 is that a section of BIOS contains tables that are used to describe each physical device in the system. Thus any number and/or type of mass storage device could be utilized as long as

hardware and software interfacing is implemented for each device in the BIOS. The following mass storage list is feasible with CP/M 2.0:

LOGICAL	PHYSICAL	APPROX CAPACITY IN BYTES
A:	Double density floppy disk 0	500k
B:	Double density floppy disk 1	500k
C:	Double density 5 inch floppy	150k
D:	Hard disk	20meg
E:	Single density floppy disk 0	256k
F:	Single density floppy disk 1	256k

In the above list, there is an example of one physical device, floppy disk 0, having two logical names, A: and E:. This was done because dual density floppy disk controllers can read/write in either single or double density. This implementation gives a means for easily transfering information from single to double density or vice versa.

When using any version of CP/M, disk drives are logged-in at the CCP by simply typing the logical name followed by a colon and carriage return (cr). In the above system, to log-in floppy disk 0 in single density mode the following is typed:

```
A>E:    User types E: (cr)
E:      System response
```

When naming files, the logical device where the file is located is indicated by placing the device name in front of the file name:

```
B:STAT.COM      File STAT.COM on device B:
```

If the logical device is not given, then the logged-in device is used.

In this article, I will limit the discussion of other I/O devices to just the console (logical-CON:) and the hardcopy device (logical-LST:). When I discuss user implementation of BIOS functions and advanced uses of the STAT and PIP utility program, then I will describe other physical-logical device pairings available in CP/M.

In order to determine which files have fcb (file control block) entries in the directory, the DIR command is used. In ver. 1.4 typing DIR(cr) will give a listing of all the files that have fcbs. In ver 1.4 these files are simply listed in order vertically on the console device. In version 2.0, however, file names are listed in rows of 4 names on the console. By using file names, wild card functions, and logical device names, the following command string variations are possible:

```
DIR TEST.COM    Find and list file name
DIR B:DDT.*     List all files on device B: with primary
                name DDT
DIR *.??M       List all files that have M as last
                character of secondary name
DIR E:          List all files on E:
DIR A???.COM    Find COM files with primary name of 4
                characters with A as first character
```

In naming files, remember that secondary names are not necessary, but primary names are. Also, one space is used between names and commands. The prompt, NO FILE, is printed when the DIR command does not find a file or group of files. Finally, ver. 2.0 allows the user to designate files as SYS (System) files so that when the DIR function is given, they will not be listed in the directory. The ability to implement this option is a function of the STAT utility program and will be discussed later.

The TYPE function will read a specified file from a disk and print it on the console device. Since console devices interpret information sent to them as ASCII data, only ASCII format files will give proper print although any file type can be used. This function will read and print an entire file up to the EOF (End of file) delimiter which is cntr-z (1Ah) in CP/M. Wild card functions are not permitted. Typing a 'space' while a file is being listed will abort the TYPE function and return control to the CCP. This is also true of the DIR command.

The REN function is used to change the name of a file. The command syntax is:

```
REN HELLO.COM=TEST.ASM
```

In this case, file name TEST.COM is changed to HELLO.COM. Wild card functions are not allowed.

The ERA function is used to erase fcb entries in the directory on a disk. The data itself is not erased but the space that it occupies on the disk may be used when other files are created at a later time. If a fcb is removed, it is normally impossible to retrieve the data unless directory information is stored elswhere. In a later article I will discuss deciphering fcb information so that the user can reconstruct files when directory entries are lost. The ERA function uses wild cards so the following variations are possible:

```
ERA *.ASM       Erase all ASM files
ERA C:DUMP.COM  Erase file on device C:
ERA TEST?.*     Erase all TEST files with extra
                character in primary name
ERA *.*         Erase all files.
```

When using the *.* file name, the CCP will ask for verification by typing 'ALL FILES (Y OR N)?' in which case the user has to type Y for the function to occur. Any other character causes the function to abort.

The SAVE command is used to store an image of memory starting at location 100h, start of TPA (Transient Program Area), as a COM file. Article I in this series contains a description of the TPA. Although the beginning location of the data to be saved is always 100h, the user signifies the size of the memory image.

CP/M uses three terms that signify differing amounts of memory. The record as described in previous articles is given as 128 (80h) bytes and is equal to the size of a single sector on a single density

floppy disk. A page of memory is equal to 256 (100h) bytes and is thus two records in length. Remembering that location 00h is a position, the first page of memory is from 00-FFh, the second page is from 100-200h and so on. Thus in a computer whose address bus is 16 bits, (2 bytes), each page is addressed by all of 8 bit combinations of the lower byte with one value of the upper byte. Thus there are 256 pages in a 16 bit machine. The term block is used to describe 2 records or 256 bytes of data. Since block and page in this context have the same value, it is important to remember that page refers to memory addresses but block refers to an amount of data. Page almost always is equal to 256 but block as well as record can have other sizes when working with different operating systems. A final point is that when dealing with data in these sizes as determined by hardware, the user is working with physical concepts. Records, pages, and/or blocks can take on differing values when one is dealing in logical concepts. For example, a record in a data base system could be made up of a person's name, his/her pay scale, and address. This logical unit may need one or more records of physical space on disk.

The syntax of the SAVE command is as follows:

SAVE 12 D:HELP.TEX

In this case 12 is the number of blocks that are to be saved, and is entered in decimal values. The user has to convert hexadecimal locations into decimal blocks. Only an even number of sectors are used, so there will be times when even though one sector of data needs to be saved, the file will be 2 sectors long. Actually this does not prove to be wasteful of disk space, because as discussed in Article II, the smallest unit that can be handled by BDOS is a cluster of 8 sectors or 400h (1024) bytes. When working with hexadecimal addresses, conversion from memory locations to blocks of memory in decimal can be accomplished using the following steps:

1: Round the final address in the memory to the next highest page value. (xx00h)

2: Subtract 100h. Page 0, 00-FFh, is not saved.

3. Convert the most signficant nibble to decimal and then multiply by 16 (16 pages in 1000h).

4. Convert the second most significant nibble to decimal and then add to value computed in 3.

5. The result is number of pages needed to save memory image.

Here is example of a memory image from 100h to 2E6Ah:

1: 2E6Ah := 2F00h
2: 2F00h − 100h := 2E00h
3: 2h := 2 dec, 2 * 16 = 32
4: Eh := 14 dec, 14 + 32 = 46
5: 46 pages is the size of memory image

When using the SAVE function for files longer than 16k bytes, areas of the TPA will be destroyed when using CP/M ver. 1.4 because the CCP uses this area when building extension file control blocks (See Article II). Thus only one SAVE can safely be done. CP/M 2.0 uses areas outside the TPA for this function allowing multiple saves of the same memory image.

The final built-in command of the CCP is the LOAD file and execute function. This function is implemented by simply typing in the primary name of the file to be loaded and then a carriage return. Only COM files will work and any other file type will generate an error prompt and the system will return to the CCP. The file is loaded at 100h and then the computer jumps to this location. Programs that are run can have differing interactions with CP/M depending on their coding. Programs can be totally independent or they can use functions and subroutines available in BDOS and BIOS via a group of SYSTEM calls. These functions will be the topics of subsequent articles on CP/M. Also the term transient program is often used for files as loaded and executed in the TPA.

A function found only in CP/M 2.0 is USER. With this command, the operator can specify a user number of 0 to 15. The result of this is that only files as previously stored under that number can be accessed by the operator. Thus all the CCP commands are effected. When the system is initially booted up, the user number is 0 which is where files stored under ver 1.4 are found. To change the user number the following is typed: USER <0-15>. To copy files from one area to another, the PIP 2.0 utility is needed although the SAV and USER functions can be used with memory images. Last of all, the function ERA *.* will not erase the entire directory in ver 2.0; the quickest way to erase the disk is to use a utility such as a disk format program that clears all sectors.

All input of the console is buffered in the 128 bytes of memory from 80h to FFh as is disk I/O when the system is at the CCP level. After a program is loaded, the CCP will save all the information in the command line excluding the original entry.

RUN TEST EMPTY.BAS $L HEX

would be stored as:

TEST EMPTY.BAS $L HEX

beginning at 81h with the number of characters (21) being stored at 80h.

The transient program can read up to 128 characters of information from this area using string handling routines. Also, the second entry (TEST in the example) is place at the default fcb location tfcb (5Ch) while the third entry (EMPTY.BAS) is placed at tfcb + 16 (6Ch). Since the full fcb is 33 bytes long, the user program must move the second file name. The use of these functions will also be discussed with system calls in a future article.

CCP CONTROL CODE OPERATION

Since console I/O is buffered, the user can edit text strings by typing control characters. The carriage return code instructs the CCP to execute the command string typed in just previous to it. If a (cr) is typed when no other information has been input, then the disk prompt is printed. Control codes are selected on the keyboard of the console by first depressing the control key and then the desired character. Certain keyboards have function keys that are substitutes for control codes. The control key functions by forcing bit 6 (40h) of the alphanumeric key depressed to zero, thus only those codes that have bit 6 set (1) will be effected:

CHARACTER	ASCII CODE	CONTROL CODE	FUNCTION
M	100 1101	000 1101	CARRIAGE RET
J	100 1010	000 1010	LINE FEED
H	100 1000	000 1000	BACK-SPACE
I	100 1001	000 1001	TAB

The codes used by the CCP are shown in Table 2:

TABLE 2 - CCP CODES

CHARACTER	FUNCTION KEY	ASCII CODE	FUNCTION
ctl-U		15h	delete line from buffer but do not erase from console screen; # is printed at end old line to indicated deleted line
ctl-X		18h	same as ctl-U but erases line from screen
	RUBOUT (RUB) DELETE (DEL)	7Fh	delete last character in the console buffer but echo it on screen (command string is typed backwords as DEL is depressed
ctl-H	BACK-SPACE	08h	same as rubout but last character is deleted from screen implemented as CCP function in ver 2.0; user option installed in BIOS in ver 1.4
ctl-R		12h	retype console buffer; used with DEL to give clear display of string; # is printed at console at end of old line before printing to indicate deleted text
ctl-E		05h	breaks line at console by sending (cr)(lf) to console without entering (cr) in console buffer; allows line of up to 128 characters to entered on console that allows lines of shorter length
ctl-M	CR, RET RETURN	0Dh	(cr)(lf) sent to console then command string is interpreted and executed by CCP
ctl-J	LINE FEED LF	0Ah	same as ctl-M
ctl-C		03h	CP/M system reboot (see discusion below)
ctl-Z		1Bh	not a CCP function; used to indicate end of console input in utility programs
ctl-S		13h	used to stop printout to console during DIR, TYPE, or similar functions in transient programs; typing any key will cancel ctl-S
ctl-P		10h	text printed on console device will also be printed on list device; if function is active then ctl-P cancels effect

While in CCP mode, inputing ctl-C causes a 'warm-boot'. When this occurs, CP/M executes a routine in BIOS that brings in the CCP and BDOS. If implemented while in CCP mode, the net effect is that the system logs in device A: and is ready to begin operation as if the system was initially booted on power up. Many transient programs implement a ctl-C option to return to CCP mode so care must be used not to execute this function accidently causing a loss of work and/or data. Also, when programs return control to CP/M, they usually do so by jumping to location 0 or by using the reset system call of BDOS which directs the computer via jumps to the warm boot routine in BIOS. When the warm boot function occurs or when a new device is logged-in for the first time after a warm boot, the disk is checked for read/write status. Using the STAT utility, disks can be software protected, and the CCP can also tell when a disk has been placed in a drive that has been initialized with another disk. As a result of both software write protection or swapping of disks, an error code will be generated when data is written to the disk. Thus whenever changing disks a ctl-C must be typed. Also, a warm boot will not change the contents of the TPA so that programs that have been developed using one disk can be saved after swapping disks in the same drive. When the CCP

cannot alter disk contents because of write protection then the following statement is printed on the console:

BDOS ERROR ON A: R/O

A can be any logical device and R/O means Read Only.

ONE, TWO or THREE DRIVES?

Many computer users when first researching mass storage alternatives ask the quetion: 'How many drives are needed for my application?' Although alternatives can vary depending on application, my experiences have given the following conclusions. First of all, the two drive system is the minimal configuration for intensive work. As mentioned above, file duplication on different disks is a necessity for protection against loss of data, but even though this can be done with one drive, it can be quite time consuming. The PIP (Peripheral Interchange Utility) is used to copy files from one disk to another. In one drive systems, two different floppy diskettes can be used by swapping disks when required by the system. When the system requires a change of disk, it will print the command 'MOUNT B:' or 'MOUNT A:' depending on whether information is to be read from A: or written to B:. This procedure can be very confusing, and can be costly when copying original files and errors occur. It should be noted that this facility is implemented in BIOS, and it may or may not be present depending on the BIOS in the system. Also, some BIOS' have this function as an option during assembly of the BIOS source code while other systems use the prompt during system boot up of: 'HOW MANY DISK DRIVES?'. With two or more storage devices, however, file duplication using PIP is a simple chore.

Probably the best configuration in terms of number of units is three. One of the areas needing more development is multi-tasking software. Multi-tasking hardware/software systems have the abiltity to perform two or more functions at the same time. This is accomplished through procedures that allow routines to share computer time. Several programs have been developed that use multi-tasking, and for the most part, these have been based on SPOOL or DESPOOL functions. In the early days of computing, when computers could only accomplish one task at a time, having the computer spend time printing information on list device or entering data from card readers could be both expensive and/or problematical due to scheduling considerations. A simple solution was to write (SPOOL) the information to be printed on a mass storage device which usually was magnetic tape; hence the term SPOOL. At a later time, the information could be printed (DESPOOLED) onto a printer which was either on-line (connected to and controlled by the original computer) or off-line (not connected to the original computer).

In CP/M programs, time that is spent while the computer waits for input from the console is used to output information on a disk file to the list device. This can prove to be a great time saver in installations that require a lot of printing. One problem, however, is that the disk containing the file that is being printed cannot be removed from its drive until completion of despooling. With a two drive system, this causes problems if two disks are required for an operation, for even though space on the despooling disk can be used, the non-despooling disk is the only free disk. With the three drive system, one drive can be dedicated as in the above example while two drives are left free.

A second advantage of having three drives is that one of the drives can be write-protected while the other two are free for both reading and/or writing. This allows the user to protect important files from possible loss due to mistake or accident. Another point is that one drive can be dedicated to holding the system diskette and various utilities while the other two are free for disk swapping.

A final advantage, and in my mind the most important, is hardware backup. In situations where the computer is a necessity for operation, failure of hardware can prove disasterous, and due to this, entire computer manufactering firms have been built or broken by the ability of users to get quick and effective maintenance. At the present time, this is by far the biggest problem in the microcomputer industry. Although microcomputers have proven to be very reliable, many tales have been circulating about failures of equipment and days, weeks, and even months of computer 'down' time. Since the disk unit is a device with moving parts that can wear out or lose adjustment, it is one of the first devices to fail and due to its nature one of the most difficult to repair. With the three drive system, if one drive malfunctions, then the other two are still available while the third is off-line. In most cases, the user will not need to alter hardware except in that case where drive-0 (the SYSTEM drive) is effected.

WHICH DISK SIZE, TYPE & DENSITY?

Another question commonly asked is: 'What size, type, and/or density format do I need?' My opinion on type of drive for most micro-computer installations, at the start, is 8 inch single density format. The reason is that this is the most time proven and standard media for microcomputing. Other systems such as tape, hard disk, and even 5.25 inch floppy disk although viable have problems due to price, availability, capacity, and most importantly, dependability. The reason I maintain single density is that the standard in the industry for the transferring of data is still single density. Although the bugs seem to have been worked out of double density hardware/software in the 8 inch drive, I suggest than when purchasing or updating to this type system, that it be thorougly tested before purchase and use. Users should also beware that many disk drives are rated for both single and double density use, so when purchasing a single density system, check the drives so that update to dual density at a later time can be done without change of drives, the most expensive component. Another consideration is that when purchasing dual density systems, (can perform single and

double density operations), check the software and documentaion for clearness and ease of single vs. double density operation. Although 5.25 inch disks have proven dependable, cost effective, and advantageous over larger devices in physical size and weight, they have been used mostly in microcomputers or stand-alone devices such as smart terminals or word processors. The 8 inch variety has been used widely in the entire computer industry, and when disk formats are standardized for the interchange of data between different systems, the 8 inch disk will probably be used.

HARD DISK SYSTEMS

Small, high capacity, cost effective hard disk alternatives have developed quickly over the last year. Also, S-100 controllers have appeared for older hard disk designs. Capacities range from 5 megabyte on up for single units with multi unit sytems controlled by CP/M 2.0 getting into the 100 megabyte range. Of importance to the average CP/M user is the fixed disk alternatives that are becoming competitive with floppy disks. Some floppy disk manufactures are building units that are hard disks within 8 inch floppy disk housings, have similar if not identical signal connections, and have the same power requirements as their flexible counterpart. As a result of this, the new idea is to mix hard disks with floppies using one controller and CP/M 2.0 software.

There are two reasons why these disks are cost effective, smaller, and more energy efficient. One, Winchester Technology, allows very high densities of data per track and tracks per disk. Secondly and most important to CP/M users is that the storage medium is non-removable. This allows the manufacturer a lot more mechanical freedom than in systems where movement of the disk due to physical support becomes a problem. As a result, these new 8 inch hard disks although offering large capacity do not offer disk backup. As long as the user does not use up his/her disk space, need to transfer data on mass storage media, need to get new data onto his/her disk systems, or have an accident, hard disks are fine.

In other words, unless the media is removable, having a second floppy is a necessity. Even if all or part of the media is removable, CP/M software will still be distributed on 8 or 5.25 inch floppy unless the software distributor has hardware that is identical to the user's. The real value of the hard disk is in using its storage capacity to greatly expand computer memory. Since data transfer on hard disks is much faster than floppies and much larger files can be maintained, operations such as searching and sorting or storage and retrieval of system memory images become quite feasible on 8 bit and 16 bit (8086 or Z8000) CP/M systems. When backup storage on floppy disk becomes a problem due file length, then magtape units based on digital cartridges become a feasible alternative and as disk technology develops, this area will also expand.

IN CONCLUSION

A few final remarks. If you are new to the mass storage market, do not be afraid to buy now for fear that your purchase will quickly become obsolete. Try to buy equipment with the philosophy that if expansion is needed at a later date, then hardware should be supplemented rather than replaced. Microcomputer equipment is like stereo equipment: once purchased its resale value drops quickly, thus replacement can prove quite costly. As far as obsolesence is concerned, as long S-100 bus systems are used, the user has a world of manufacturers and products to draw from. If one device needs to be replaced, the entire system need not be replaced. This philosophy is quite unique to the S-100 industry for a great majority of manufactures still viable today have survived because they have used industry compatibility as a major marketing point. The same can be said of CP/M and CP/M compatible operating systems.

An Introduction to CP/M — Part 4

Jake Epstein

Utilities and BIOS

In the previous articles in this series on the CP/M operating system, the focus has been on general concepts. With the first part of this article, I will conclude discussing introductory material and begin exploring more technical matters. Since I will be dealing with assembly language, those readers who have not had experience in this area might find it helpul to refer to one or more of the references listed at the end of this article.

CP/M Utilities

Digital Research supplies several utilities with CP/M. These various programs are used for file management, text processing, program development, and information transferral. Rather than discuss each program in depth, I will merely list them and give a brief description of their function. Certain utilities will be covered in depth in following articles when system alteration and program development are discussed. If anyone is interested in submitting articles devoted exclusively to the description and use of specific utilities and/or other CP/M programs, feel free to contact S-100 Microsystems.

ED.COM - This is a line oriented text editor used for preparation of ASCII files. With CP/M, it is usually used in co-ordination with ASM.COM and SUBMIT.COM, but can be also used to prepare text for other programs such as BASIC or a TEXT PROCESSOR.

ASM.COM - This program is used to assemble Intel 8080 standard format ASCII files. Input files have the secondary name ASM while output files have secondary names of HEX and/or PRN. HEX files, which are in Intel hex format, contain a representation of the machine code of the assembled source program. PRN files are identical to the ASM source file but with machine code equivalents listed in hexadecimal code adjacent to each line of assembly code.

LOAD.COM - Hex format files are converted to executable binary files using this program. The output file produced is a COM file which can then be executed using the CCP LOAD and EXECUTE command.

STAT.COM - File characteristics such as length or write protection are checked with STAT. Logical devices can also be checked using this program. The STAT utility is much more comprehensive in CP/M 2.0 and later versions than in earlier versions.

DDT.COM - This program provides an extensive array of commands that are used to inspect and modify the contents of system memory. It can be used to create, analyze, and debug programs with the following functions: 1. alter memory using hexadecimal numbers or Intel standard assembler mnemonics. 2. disassemble machine code to Intel Standard mnemonics; 3. dump areas of memory showing both hexadecimal equivalents and ASCII equivalents of each location; 4. execute a program with breakpoints; 5. list and alter the contents of processor registers and flags; 6. perform hexadecimal addition and subtraction; 7. perform single or mutiple step execution of programs showing machine register contents and flag status after each instruction; 8. load files from disk to any location in memory.

SUBMIT.COM - This program is used for batch processing. Using ED.COM, the user prepares a file that contains a list of CCP commands to be executed by the operating system. This file has the secondary name SUB. When SUBMIT is invoked, all the commands listed in the SUB file will be executed just as if they had been entered at the console. SUBMIT will only work when drive 0 (system drive) is logged in. In CP/M 1.4 and earlier versions, once the user leaves the CCP mode, as when executing a program, the SUBMIT function relinquishes control of the system until a warm boot (cntr-C) is executed by the user. In version 2.0 and later, commands that are interpreted by a program can be programmed using the XSUB.COM utility in co-ordination with SUBMIT. Thus, long and complex processes involving many different devices and programs can be executed with or without user interaction.

MOVCPM.COM - System alteration for various sized memories is accomplished using this utility. MOVCPM will be discussed in depth under system alteration in a future article in this series.

PIP.COM - The peripheral Interchange Program is used to transfer information from one location to another. The locations involved can be disk files or I/O devices such as the console or the printer. PIP also contains several software switches that allow for verification or alteration of the data flow.

SYSGEN.COM - This program is used to read or write to the system tracks of a diskette. This allows the operator to alter components of the operating system. This utility will also be discussed in depth under system alteration in a future article.

In addition to the COM files listed above, Digital Research provides two ASCII files that are used in alteration of a 2.0 or later version system. These files, DISKDEF.LIB and DEBLOCK.ASM, will be dicussed in

the next article under BIOS II. The term TRANSIENT COMMAND is often used for utilities such as PIP and STAT because they are often used in a fashion similar to CCP commands to monitor and alter system status. Rather than being built into the CCP, however, they are loaded and then executed in the TPA, Transient Program Area.

BIOS - Part I

As discussed in the first article of this series, BIOS (Basic Input/Output System) is the the module of CP/M in which software interfaces to computer peripherals are located. The BIOS that is provided by Digital Research is for the Intel MDS INTELLEC computer system. If the user does not have this computer system, then the BIOS has to be modified for his/her hardware configuration. Once implemented however, software that is CP/M- compatible may be executed without need for additional modification of the operating system. Manufacturers of disk controller boards for 8080/Z80/8085-based microcomputers supply BIOS's, but the user may still have to modify non-disk routines such as serial or parallel port drivers. The main focus of this section and BIOS - Part II will be on the structure and operation of BIOS with a few examples of modification. This section will be devoted to general organization of BIOS and non-disk I/O routines. Part II will deal with disk I/O routines with emphasis on version 2.0 enhancements. It is advisable that the reader obtain a listing of a working BIOS to use as a reference to this section. Issue number 2 of S-100 MICROSYSTEMS contains an excellent BIOS for version 1.4 written by Martin Nichols (see references).

Structure

BIOS is the last module in the CP/M memory map. All versions of CP/M have the following routines:

1: COLD BOOT	This routine is used to initialize various areas of the operating system after the CP/M is loaded from the system diskette.
2: WARM BOOT	Used after a system reset (cntr-C) to load in CCP and BDOS without the need to load and initialize the entire system.
3: CONSOLE STATUS	Used to determine whether or not data is ready to be input from the console device.
4: CONSOLE INPUT	Used to input data from the console.
5: CONSOLE OUTPUT	Used to transmit data to the console device.
6: LIST OUTPUT	Used to transmit data to the list (hard- copy) device.
7: PUNCH	Used to transmit data to the paper tape punch.
8: READER	Used to input data from the paper tape reader.
9: HOME	Causes the logged in disk drive to seek track 00.
10: SELECT DISK	Used to select the disk drive where disk I/O will take place. Used to log in a disk.
11: SET TRACK	Initializes the disk controller to a specified track.
12: SET SECTOR	Initializes the disk controller to a specified sector.
13: SET DMA ADDRESS	Sets the beginning of a buffer area in system memory where data transferral will occur during disk I/O.
14: READ	Data will be transferred from the disk to the DMA buffer as determined by the above routines. The amount of data is that which is contained in one physical unit on the disk - one sector of 128 bytes on a single density floppy diskette.
14: WRITE	Same as read only data is transferred from the DMA buffer to the disk.

The following two routines have been added to CP/M 2.0 and later versions:

15: LIST STATUS	Used to check status of the list device.
16: SECTOR	Used to convert from a logical location TRANSLATE to a physical location of a sector. Used with translation tables found elsewhere in BIOS. See Article II in this series for a preliminary discussion of sector skew.

At the beginning of BIOS, a vector jump table is located to direct programs to the various routines. The various jumps have to be placed in the order given above, but the routines themselves may be anywhere. There are three different ways in which user programs may interface to I/O devices. 1. The actual software drivers may be part of the program. In this case the program is not useable in systems different from the development system without modification. 2. The program may use system calls to BDOS to accomplish I/O. In this case, BDOS and BIOS transfer data as determined by one of 36 numbers placed in general microprocessor register 'C'. 3. A final possibility is in direct calls to the appropriate routine in BIOS. This is useful when a desired I/O function is not implemented in BDOS. An example of this is console input. In CP/M 1.4 and earlier versions, any input via a console input system call will cause the data to be transmitted back to the screen. In order to eliminate this character echo, one merely needs to call the fourth jump vector, console input, of the BIOS jump table. One disadvantage to this procedure is that the calls from the user program must be done in a manner that is compatible with different sized CP/M systems. See

listing 1 for an example of how this can be accomplished. A final note; additional routines may be added to BIOS, but jump vectors must be placed at the end of the jump table so that its continuity is not upset.

Besides the above routines, various implementations may have areas for temporary storage or data buffering. In CP/M 2.0 and later versions, areas of BIOS are reserved for disk information. Each disk unit in the system has a disk parameter block and a sector translation table stored in BIOS. This gives the operating system the ability to handle diferent types of mass storage units. The disk parameter block contains information such as disk size, sector size, sectors per track, etc. The sector translation table is used to determine sector skew. These areas are used by BDOS to calculate physical sector locations from logical file information. Also, BIOS contains a buffer area for directory operations. These topics will be discussed at length in BIOS - PART II.

At this point I will go into more depth on the CP/M memory map. This information is important in understanding some of the parameters influencing the location and size of the BIOS. It will also serve to introduce material that will be expanded in the article dealing with system alteration. The following table shows the size and relative locations in a minimal-size CP/M memory map.

MODULE	SECTORS	STARTING ADDRESS	SIZE IN BYTES	
(Map of 16K CP/M 1.4 System)				
---	--	00H	100H =	256
TPA	--	100H	2800H =	10240
CCP	16	2900H	800H =	2048
BDOS	26	3100H	0D00H =	3328
BIOS	4	3E00H	200H =	512
Totals	46		4000H =	16384
(Map of 20K CP/M 2.0 System)				
---	--	00H	100H =	56
TPA	--	100H	3300H =	13056
CCP	16	3400H	800H =	2048
BDOS	28	3C00H	E00H =	3584
BIOS	7	4A00H	380H =	896
Totals	51		4C80H =	19840

The minimal memory size configuration available in CP/M is 16K in 1.4 and earlier versions and 20K in CP/M 2.0 and later versions. For larger size-configurations, the CCP and BDOS have to be relocated using MOVCPM, while BIOS and the COLD START LOADER have to modified and reassembled to the new system memory size. Although the CCP, BDOS, and BIOS remain in the same relative locations, the TPA is either expanded or contracted to fill out added or deleted space in various sized systems. In the tables, the TPA begins at 100H and is either 2800H or 3300H long depending on version. Adding 100H to either of these values gives the staring point of the CCP: 2900H in 1.4 and 3400H in 2.0. To calculate where the CCP would start in other sized systems, simply use the following formula:

FCCP Start (CBASE) = TPA size + (BIAS * 400H)
where: BIAS = Memory size - Minimal configuration (in K bytes)
400H = 1024 or 1K bytes

BIAS is actually the number of 1K segments that the memory is greater than the minimal configuration. The following equations should clarify this material.

For a 24K 1.4 System:

BIAS = 24K-16K=8K
CCP START = 2900H + (8*400H) = 2900H + 2000H = 4900H

For a 62K 2.0 System:

BIAS = 62K - 20K = 42K
CCP START = 3400H + (42*400H) = 3400H + A800H = DC00H

To calculate the location of BIOS simply add the combined size of BDOS + CCP to the CCP starting location. The following tables give the CCP and BIOS starting points for various systems.

16K	0	2900H	3E00H			
17K	1	2D00H	4200H			
18K	2	3100H	4600H			
19K	3	3500H	4A00H			
20K	4	3900H	4E00H	:	0	3400H 4A00H
21K	5	3D00H	5200H	:	1	3800H 4E00H
22K	6	4100H	5600H	:	2	3A00H 5200H
23K	7	4500H	5A00H	:	3	3E00H 5800H
24K	8	4900H	5E00H	:	4	4400H 5A00H
28K	12	5900H	6E00H	:	8	5400H 6A00H
32K	16	6900H	7E00H	:	12	6400H 7A00H
40K	24	8900H	9E00H	:	20	8400H 9A00H
48K	32	A900H	BE00H	:	28	A400H BA00H
56K	40	C900H	DE00H	:	36	C400H DA00H
64K	48	E900H	FE00H	:	44	E400H FA00H

A problem can arise when dealing with different implementations of BIOS. Notice that in a 1.4 system there are only 512 bytes of memory, xE00H-xFFFH, available for BIOS. Since most BIOS modules will be larger in size, especially when long drivers such as video routines are added, certain functions will have to be placed in ROM and then called from BIOS. Since the system diskette has a total of 9 sectors available -- the total number on tracks 0 and 1 minus the total needed by the COLD START Loader, CCP, and BDOS -- BIOS can be a total of 9 * 128 or 1152 bytes long To build a working system, BIOS could start at location xA00H. To do this, the system would be built one K smaller than avaialble memory. For example, for a system with 24K of available memory, a 23k CP/M system would be built using the MOVCPM utility, and BIOS and the COLD START LOADER would be reassembled for the proper CCP, BDOS, and BIOS starting points. To complete the modification, the number of sectors read by the COLD START LOADER

and the WARM BOOT routine in BIOS would have to be adjusted. Often , distributors of CP/M systems may alter MOVCPM so that this 1K offset is handled automatically. Problems arise when a BIOS that has been built for the standard MOVCPM is added to one of these systems. The easiest solution to the problem is to simply build system a 1K smaller, and then proceed with the modification of CP/M in the normal manner.

In CP/M 2.0, the limitation is not in system memory size but in disk space. There are only 7 sectors (380H bytes) available on a single-density 8-inch disk for BIOS, and although this is enough storage for a minimal BIOS, any additions will quickly overflow this space. One way to get around the problem is to place all buffer and scratch RAM areas at the end of BIOS. Thus only permanent code need be loaded. If certain RAM locations need to be initialized, then routines will have to be placed in the COLD BOOT module of BIOS to accomplish this. This solution could also make the permanent code larger than 7 sectors. Another option is the placing of certain routines in ROM. The easiest and most common method would be to place the ROM in the last area of memory, usually F000H or F800H. Since most routines will need stack space and scratch areas, it is wise to have RAM located in this area also. Some of the newer ROM boards have space for RAM, but if this option isn't available, then space will have to be reserved in an area outside the bounds of CP/M in system memory. A problem with this configuration is that any time system memory is added and/or CP/M is enlarged, the ROM may have to be erased and reburned if it is an EPROM; or a brand new ROM purchased if it is not. There are ways of computing locations using elaborate routines in software, but the easiest way is the one using a ROM/RAM board. Note: the Z80 and the new 16 bit microprocessors (Z8000, 8086, 68000) do provide relative and indexed addressing schemes that would help circumvent this problem.

Another solution is in configuring the system so that there are more than 2 system tracks. This can be done easily in CP/M 2.0 by using the DISKDEF.LIB file and the CP/M MACRO Assembler. I will discuss this in BIOS-PART II. A problem with this is that software supplied from other sources may not be compatible with the modified system. To solve this problem, having a logical device that can read standard diskettes could be implemented in BIOS. As an example, in a two-drive system, devices A: and B: could be interfaced to drives 0 and 1 with their configuration being the one most commonly used in the system. Logical device C:, on the other hand, could be set to access drive 1 with its configuration being different than A: and B:. Using PIP, software could then be transferred between A: and C:.

A final method would be to have a minimal BIOS stored on the system tracks with an expanded BIOS stored as a file. Upon COLD START, this file could be ovelaid in memory either manually or automatically. CP/M 1.4 and 2.0 can be modified so that a file is loaded and executed upon COLD BOOT. I will discuss this option in the article on system calls.

BIOS - Modules

The following discussion decribes the parameters of non-disk I/O modules in depth.

CONSOLE STATUS: Upon return from this routine, the value 00 in register 'A' indicates that no character has been input at the console device; 0FFH indicates a character has been input.

CONSOLE INPUT: Upon return, register 'A' will contain an input character. The most significant bit (parity bit) should be reset to zero. This is accomplished most easily with the operation ANI 7FH. Routines may be added to convert characters to different values. For example: delete (7FH) backspace (08H).

CONSOLE OUTPUT: Upon entry, register 'C' contains the character to be printed. The 'A' register will be changed except where Z80 instructions are used. Routines may be placed to control output.

READER: Same parameters as CONSOLE INPUT. May be modified to function with other devices such as modems for telephone communication.

PUNCH: Same parameters as CONSOLE OUTPUT.

LIST: Same parameters as CONSOLE OUTPUT. Routines can be added to control printer paging to or route (SPOOL) data to a disk file.

LIST STATUS: Same as CONSOLE STATUS. Used with SPOOL or DESPOOL programs.

Programmed I/O

Martin Nichols' BIOS serves as an excellent example of how to implement the above routines. In all cases, there are four possible ways of programming these routines. The most common and easiest approach is polled I/O. In this method, I/O is accomplished only when the routines are called. In order for each routine to operate properly, status has to be checked for each device, with program loops being used to force the computer to wait until the device is ready for I/O. In other words, the routine polls the I/O device's status register to determine when it is ready to send or receive data. In most commercially prepared BIOS's, options that can be selected at assembly time are given for various boards. If a board is not covered, then the user must program into the appropriate routine the location and the logic of the status register and the location of the DATA register. A BIOS routine that allows no I/O usually indicates an error in port location. A quick stream of characters with only one input or output usually implies improper status logic or bit location. The following routine shows how to program an I/O port:

```
CONOT:                  ;CONSOLE OUT ROUTINE
        IN      00      ;INPUT STATUS FROM PORT 0
        ANI     02      ;CHECK BIT BY ANDING 'A' WITH 0000 0001B
        JZ      CONOT   ;IF 'A' REGISTER WAS 0000 0000B THEN LOOP
                        ;IF NOT GO ON (JZ MEANS JUMP IF ZERO)
        MOV     A,C     ;PUT CHARACTER IN 'A' REGISTER
        OUT     01      ;OUTPUT CHARACTER IN 'A' TO PORT 1
        RET             ;RETURN TO THE CALLING PROGRAM
```

In this routine, the logic is positive because the status bit must go from 0 to 1 to indicate that the device is ready. Negative logic is that case where the bit changes in a negative direction (1 to 0). In a negative logic system, the third instruction would be JNZ (JUMP if not zero).

A second method is memory mapped-I/O. This technique is almost identical to polled I/O except that memory access instructions are used instead of inputs and outputs. Although programming using this method can be quite efficient, a major disadvantage is that I/O ports use system memory space. An example of a memory mapped I/O scheme is the keyboard of the Radio Shack TRS-80 microcomputer.

Interrupts

Another method is interrupt-driven I/O. In this case, when a device is ready to perform I/O, it interrupts the computer from its current task so that it (the computer) can perform the operation. This eliminates the need for the computer to sit and wait while it checks status via a loop as in the above example. Also, interrupts can be used to control program execution. Suppose that you have written a program that outputs integers to the screen. If it is written as an infinite loop, then the only way to halt execution may be a system reset. If, as a part of an interrupt-driven I/O routine, input is checked for a certain character such as ESCape (1BH), program execution could be broken by pressing the appropriate key. One of the main uses of the CONSOLE STATUS routine is for the same type of function.

The main problem with interrupts are that they are much harder to implement than polled I/O. This is true for both hardware and software. To understand what is involved, I will devote a bit of discussion to the hardware and software interfacing of the Intel 8080 microproccesor. I suggest consulting the Intel 8080 Users manual and the articles on the proposed IEEE S-100 standard that are listed under the reference section at the end of this article for diagrams, timing charts, and descriptions.

Before an interrupt can occur, the 8080 must be "software- initialized" (instruction is part of a program) using the ENABLE INTERRUPT (EI - FBH) instruction. Upon initialization, a flag inside the 8080 is set to indicate that interrupts are enabled. To interrupt the computer, the I/O interface logic pulls pin 14 high on the 8080. This pin is connected to line 73 of the S-100 bus, but it is interfaced so that the I/O device has to pull it low to cause an interrupt. After this occurs, the computer will finish its current instruction and then service the interrupt. The interrupting hardware provides the next machine instruction instead of program memory as pointed to by the 'PC' (program counter) which normally is the case. This is accomplished via an 8080 status signal that indicates an interrupt read instead of a memory read. This signal is supplied by S-100 line 96, as opposed to line 47, which indicates a memory read.

Although any instruction may be supplied, the usual procedure is to use one of the 8 RST (restart) instructions. These instructions have two effects. First, they all cause the value of the 'PC' to be saved on the processor stack with the usual update of register 'SP' (the stack pointer), as is the case with a CALL instruction. Then, depending on the instruction, the program will jump to one of 8 locations: 0, 08H, 10H, 18H, 20H, 28H, 30H, 38H. With CP/M, three of these locations cannot be used. Location 00, which is restarted by instuction RST 00 - C7H, is reserved for the warm boot jump vector, as discussed in Article 3 of this series. Location 30H, RST 6 - F7H, is the first location of an 8-byte block reserved for future use by Digital Research. Finally, location 38H, RST 7 - FFH, is used by utility programs such as DDT to control and/or monitor program execution. By placing a RST 7 instruction in place of another instruction and saving the replaced instruction and its location, a program can be run using the computer itself to execute the program up to the instruction. At that time the computer breaks out of the program and control returns to the utility, which may or may not return the initially changed instructions to the program. In DDT, two RST 7 instructions may be placed in the program at one time and are called "breakpoints".

Although the RST instruction may be supplied by the I/O port interface, often a separate interface board will be used to supply the instruction. The I/O port will request an interrupt via one of 8 VECTOR INTERRUPT pins on the S-100 bus, pins 4-11. When activated, the vector interrupt interface will generate an interrupt through S-100 line 73 and then place the RST instruction on the data bus at the appropriate time. The CPU will indicate when it is ready for this transfer via line 96; this line is used in place of line 47, which indicates a memory read. Each vector interrupt pin corresponds to a specific RST instruction with pin 4 referring to RST 0, pin 5 - RST 1 and so on. The vector interrupt interface may also contain a priority encoder. This device allows the user to set up different priorities for vector interrupts. While an interrupt is being serviced, other requests may occur in systems with more than one interrupt-driven device. When the time comes, the device with the highest priority will be serviced next. In a system with more than one device, this facility is necesary to prevent timing errors. In my next article, I will include a diagram of a vector interrupt interface that I built to use interrupts with my serial I/O board.

Since the restart instruction saves the program counter, which is set to the next instruction of the interrupted program, a simple return at the end of the interrupt service routine will allow the program to resume where it left off. Of course, all CPU registers altered by the service routine have to be returned to their state prior to the interrupt. This is best done with PUSH and POP instructions. After the interrupt, the internal flag that enabled the CPU to accept an interrupt, as mentioned above is reset to zero. Thus, the interrupt service routine must reset the flag to 1 using the EI (enable interrupt) instruction. Where this instruction is placed is important. If it is placed at the beginning of the routine, then it is possible that nested interrupts could occur. In other words, an interrupt service routine could be interrupted, which also could be interrupted, and so on. If the EI instruction is placed at the very end of the routine, then a new interrupt will not be serviced until the prior routine is completed. Interrupts must be disabled during CPU controlled data transfers that must be continued to completion. In most CP/M systems, this is during the SECTOR READ or SECTOR WRITE routines when using polled

I/O boards such as the Tarbell single-density controller. If the disk controller has a data buffer (an area to hold data) then interrupts will not cause problems.

Direct Memory Access

The first three I/O methods are commonly termed programmed I/O techniques as distinguished from the fourth, DMA (Direct Memory Access). When S-100 line 74 is pulled low, 8080 pin 13 is pulled high. This causes the CPU to enter a HOLD state. During this time, the CPU relinquishes control of the computer address buss (group of lines used to transfer address information) and data bus. The I/O device then can directly transfer information to and from memory at much greater speed than is possible when the CPU is involved during standard programmed I/O. There are two general ways in which DMA can be implemented. In the first, the calling routine loads certain registers of the device with data such as sector, track, or DMA address, etc., and then issues a data transfer command. In the second method, an area of memory is loaded with this information, and the DMA controller reads this area during a DMA sequence. I will give more information on this in BIOS-PART II. While in a HOLD state, the 8080 CPU will not honor interrupts; thus, interrupt-driven I/O routines will not conflict with DMA routines. An example of a non-disk device that may use DMA is a video board that uses system memory for its refresh memory as is the case with the Processor Technology VDM-1 video board.

Sample Routines

I will now give a general outline on how to implement interrupts in CP/M. The examples that I will use will be based on the MITS 2SIO board set up at port location 10H with interrupt vector 1 (RST 1 will be used). Port 10H is the status port and 11H is the DATA port. Listing 2 shows a simple output routine using interrupts, as opposed to polled I/O, as demonstrated in the above example. This could drive a printer or a terminal. The main difficulty with output versus input is that the I/O device interrupts the computer as soon as the current operation is completed, and thus could place the system in a infinite loop between the main program and the interrupt service routine. To avoid this problem, the I/O port's output interrupt is disengaged until data is ready to be printed. Since all disk I/O routines must have DI and EI commands, the DI command would not work to disable the interrupts. A way of getting around this would be to use an interrupt status byte in memory, which would be checked at the end of each disk I/O to determine whether interrupts should be enabled using the EI command. If input interrupts are to be allowed, however, then this scheme would not work.

In practice, this output routine is no more efficient than polled I/O. Studying the routine will show that there is a built- in wait period when the computer is ready to output information but the I/O device is not ready. Memory buffers can be written into the I/O routine to greatly improve its efficiency. Listing 3 shows an input routine with a buffer. In this example, the interrupts will be disabled once the buffer is full. Thus, any data input after this state occurs will be lost until the buffer is emptied. Although there are many ways to implement a buffer management scheme, I chose to demonstrate a circular buffer using three counters and two pointers. POINT1 is the location where each byte is stored in the buffer as it is input from the I/O device. POINT2 is the location where data is read from the buffer to the main program. POS1 and POS2 are used to keep track of each pointer's position in the buffer. When the counter reaches zero (the pointer is at the end of the buffer) the pointer is set to the beginning of the buffer, and the counter is loaded with the length of the buffer in bytes. Thus the pointers move through the buffer in an endless circle.

The counter labeled COUNT represents the current amount of data waiting to be read by the computer, and is the space in number of bytes between pointers 1 and 2. When data is input from the I/O device, COUNT is incremented, and when data is read by the main program, it is decremented. COUNT is also used by the INSTAT routine to indicate to a main program when data is ready to be read. Another scheme would be to use comparison routines to indicate buffer placement and status but, in experimentation, I found these to be much longer and less efficient in terms of processing time than the method used here. Finally, listing 3 contains additional routines and was taken directly from a working BIOS that I have been using.

Character Traps

A technique that proves quite useful is character-trapping. By using 8080 CPI (compare immediate) instructions in drivers, certain characters can initiate special routines that will implement a user-defined function. In listing 3, I have placed two character traps that have proven very useful extentions to normal CCP and/or BDOS operation. By pressing the ESC key (Escape - ASCII 1BH) on my terminal, the interrupt service routine branches to a routine that awaits a second character from the terminal. In this example I have implemented just two functions, but an unlimited number could be added (with the amount of memory space available for BIOS being the only restriction). The ESC-P sequence enables output to both the console device and the list device. This is similar to the cntr-P function of CP/M, but in this case printing can be enabled or disabled in all situations and is not disabled automatically after WARM boots (cntr-C). A variation on this would be to enable other printers that have I/O drivers in BIOS. Thus, high-speed dot matrix printers versus slower wordprocessing printers could be switched on or off quite easily during execution of programs such as Microsoft BASIC-80 that allow only one list device. The second function, ESC-C, is virtually the same as cntr-C except that a warm boot can be performed at any time unless BIOS has been altered or interrupts disabled. This function is especially useful when a program "hangs-up" or the user wishes to terminate program execution and contr-C does not work. Since warm boots do not change the TPA (transient program area). ESC-C can be used instead of system reset, which does destroy

the TPA. Thus, ESC-C can be performed, and the TPA can be saved on disk. Care should be used when implementing a function such as this during disk I/O to prevent data loss. Although the ESC-P function may be used in polled I/O routines, ESC-C will not work if I/O does not occur after a program bombs.

A final technique that can be implemented via character traps is character conversion. One interfacing project that my consulting associate and I had was to interface a word processing printer to S-100 hardware and CP/M software. This printer uses escape sequences similar to ones implemented above to initialize a wide variety of features such as underlining, reverse print, and boldface (double strike). In order to initialize these functions, I incorporated a trap in the output routine to convert a printing character such as "}" (ASCII 7DH) to an ESC (ASCII 1BH). In the following example, ESC-A causes the printer to boldface print and ESC-B causes it to print normally.

DISPLAY AT TERIMINAL:
Now is the }Atime}B for all...

OUTPUT TO PRINTER
Now is the (ESC A)time(ESC B) for all...

PRINT-OUT
Now is the time for all...

When writing routines such as these, care must be used in selecting trap characters. Since traps often filter characters out of the data input stream, trapped data must not be important to application programs. For example, ESC is used by the "MICROSOFT Basic-80" line editor.

A final example of character trapping is Listing 4. This partial listing shows how to implement back space character deletion in a 1.4 BIOS. This will emulate the backspace option of CP/M 2.0.

The IOBYTE

When implemented in the BIOS, the Intel standard I/O byte function can be used to convert logical devices to specific physical devices. As an example, the console device could be a CRT terminal, a video board with separate keyboard, a printing terminal with keyboard input, or an acoustic coupler that would allow a remote terminal be the console device over the telephone lines. Although it is the responsibility of the BIOS to direct I/O to a physical device, the STAT utility or an application program can be used to modify device assignments if the BIOS is programmed to handle such functions. Location 3 of system memory is reserved for a software register labeled IOBYTE which indicates which physical device is to be assigned to a logical device. CP/M recognizes four logical devices:

1: CON:	Console	The device used by the CCP
2: LST:	List	The printer or hardcopy device
3: RDR:	Reader	Paper tape reader
4: PUN:	Punch	Paper tape punch

When BDOS directs I/O to one of these logical devices, it does so by calling a location in the BIOS vector jump table described above. The table below shows the link between logical and physical device drivers.

LOGICAL DEVICE	VECTOR JUMP TABLE
1: CON:	3: CONSOLE STATUS
	4: CONSOLE INPUT
	5: CONSOLE OUTPUT
2: LST:	6: LIST OUTPUT
3: RDR:	8: READER
4: PUN:	7: PUNCH

The IOBYTE is divided into four 2-bit segments. Each segment refers to one of the logical devices listed above. Information about physical assignments are placed in each segment in the form of binary numbers. Thus each segment can represent four different assignments. Below is a diagram of the IOBYTE showing the positional relationships of logical status information.

```
*****************************
* LST: * PUN: * RDR: * CON: *
*****************************
bit 7 6 5 4 3 2 1 0
```

The following information is the Intel standard logical to physical conversion for the four numbers stored in each segment. The user may or may not follow this standard in BIOS, but the numbers are stored in this manner when using the STAT utility to modify the IOBYTE.

1. CON:	0. TTY
	1. CRT
	2. BAT - Batch mode
	3. UC1 - User-defined console
2. RDR:	0. TTY
	1. PTR - High-speed reader
	2. UR1 - User-defined reader 1
	3. UR2 - User-defined reader 2
3. PUN:	0. TTY
	1. PTP - High-speed punch
	2. UP1 - User-defined punch
	3. UP2 - User-defined punch
4. LST:	0. TTY
	1. CRT
	2. LPT - Line printer
	3. UL1 - User-defined list

In the above list, CRT corresponds to Cathode Ray Terminal. TTY refers to ASR (automatic send/receive) printing terminals. These terminals generally have a built-in paper tape reader/punch and run at very slow speed (11 characters per second). Thus the paper tape device is referred to as a slow reader or slow punch. In contrast to the TTY punch, the fast reader or punch is a separate device that can run at relatively high speeds. For example, DIGITAL EQUIPMENT'S PC11 reader/punch will read at 300 cps and punch at 50 cps. Line printers are usually devices dedicated to hard copy (in contrast to TTYs) and have speeds that range from 30 cps to beyond

180 cps. I have used 180 cps as a general limit because true line printers which print a wholeline at a time (as opposed to character printers which do one character at a time) are really not practical for small systems. Although fast, line printers are quite large, use vast amounts of power, are expensive, and require a great deal of hardware and software interfacing.

Not all of these devices need be written into a BIOS. The following approach gives the most efficient use of space and processor time when writing IOBYTE-directed routines:

1: Write a set of drivers (input, output, input status) for each physical device to be accessed by the system.
2: Avoid including drivers that are not needed.
3: Write a linkage routine that reads and translates IOBYTE information for each position needed in the vector jump table.
4: If possible place these routines in ROM (Read Only Memory) with its own vector jump table to facilitate programming.

If this plan is followed, redundancy will be eliminated and system generation (as when updating or changing software) will be simplified. Hardware will tend to remain static, whereas software goes through a constant evolutionary process. Listing 5 gives an example of how to write a set of linkage routines for the console device. As a final note, reviewing the sections on STAT.COM and PIP.COM in the CP/M documentation will give further insight into the use of IOBYTE.

References:

"CP/M 2.0 User's Guide", (Set of 7 Manuals), Digital Research, pacific Grove, Ca.

Epstein, Jake: "An Introduction to CP/M", S-100 Microsystems, vol 1, nos. 1, 2, 3, Jan-Jul 1980.

Findley, Robert: "Scelbi 8080 Software Gourmet Guide & Cook Book", 2nd Ed., Scelbi Computer Consulting, Milford Conn., 1978.

"8080 Assembly Language Programming Manual", Intel Corp., Santa Clara, Ca, 1976.

"8080 Microcomputer Systems User's Manual", Intel Corp., Ibid.

"The IEEE S-100 Proposed Bus Standard", Reprint, S-100 Microsystems, vol. 1, no. 1, Jan/Feb, 1980.

Libes, Sol: "S-100 Bus - New Versus Old", S-100 Microsystems, vol. 1, no. 2, Mar/Apr, 1980.

Nichols, Martin: "An Improved CP/M BIOS For Tarbell Disk Controller", S-100 Microsytems, vol. 1, no. 2, Mar/Apr, 1980.

Osborne, Adam: "An Introduction to Microcomputers", vol2.-"Some Real Products", Adam Osborne and Associates, Berkely, Ca., 1976.


```
;**************
;* LISTING 1 *
;**************

;THIS PROGRAM DEMONSTRATES WRITING CALLS TO BIOS
;I/O ROUTINES THAT WILL WORK IN ANY SIZED CP/M SYSTEM
;WITHOUT MODIFICATION.

;THE TECHNIQUE USED IS TO COMPUTE THE LOCATION OF THE
;BIOS VECTOR JUMP TABLE USING THE WARM BOOT ADDRESS
;FOUND AT LOCATION 01, AND THEN COMPUTE
;THE ACTUAL LOCATION OF JUMP VECTOR USING OFFSETS.

;THE FOLLOWING TABLE GIVES THE RELATIVE POSITIONS OF THE
;BIOS VECTOR JUMP TABLE BASED ON THE WARM BOOT ADDRESS

0000 =  WARM    EQU     00      ;WARM BOOT
0003 =  CONST   EQU     WARM+3  ;CONSOLE STATUS
0006 =  CONIN   EQU     CONST+3 ;CONSOLE INPUT
0009 =  CONOUT  EQU     CONIN+3 ;CONSOLE OUTPUT
000C =  LIST    EQU     CONOUT+3 ;LIST
000F =  PUNCH   EQU     LIST+3  ;PAPER TAPE PUNCH
0012 =  READER  EQU     PUNCH+3 ;PAPER TAPE READER
0015 =  HOME    EQU     READER+3;HOME DISK
0018 =  SELECT  EQU     HOME+3  ;SELECT DISK DRIVE
001B =  SETTRK  EQU     SELECT+3;SELF TRACK
001E =  SETSEC  EQU     SETTRK+3 ;SET SECTOR
0021 =  SETDMA  EQU     SETSEC+3;SET DMA ADDRESS
0024 =  READ    EQU     SETDMA+3;READ ON SECTOR
0027 =  WRITE   EQU     READ+3  ;WRITE ONE SECTOR

0001 =  WBOOT   EQU     01      ;LOCATION OF WARM BOOT JUMP
0100 =  TPA     EQU     100H    ;BEGINNING OF TRANSIENT PROGRAM AREA

                ORG     TPA     ;LOCATION IS BEGINNING OF TPA

;THIS PROGRAM INPUTS A CHARACTER FROM THE KEYBOARD
;THEN SENDS IT TO THE LIST DEVICE. NO PRINTOUT WILL
;APPEAR ON THE CRT SCREEN.

START:
0100 310001             LXI     SP,TPA  ;SET STACK POINTER
0103 CD1201             CALL    INCHR   ;GET A CHARACTER
0106 FE03               CPI     03      ;IS CHARACTER CNTR-C
0108 CA0000             JZ      00      ;IF SO DO A WARM BOOT
010B 4F                 MOV     C,A     ;BIOS EXPECTS CHARACTER IN C
010C CD1A01             CALL    PRNCHR  ;PRINT THE CHARACTER
010F C30301             JMP     START+3 ;LOOP AVOIDING 4-BYTE INSTRUCTION

;ROUTINE TO COMPUTE LOCATION OF INPUT VECTOR JUMP
;AND THEN BRANCH TO IT

INCHR:
0112 2A0100             LHLD    WBOOT   ;LOAD THE H,L REGISTER WITH WARM BOOT
                                        ;ADDRESS IN BIOS VECTOR JUMP TABLE
0115 110600             LXI     D,CONIN ;LOAD D,E WITH POSITION OFFSET
0118 19                 DAD     D       ;COMPUTE LOCATION BY ADDING D,E TO H,L
0119 E9                 PCHL            ;PLACE CONTENTS OF H,L IN PROGRAM
                                        ;COUNTER CAUSING A JUMP
```

```
                ;ROUTINE TO COMPUTE LOCATION OF LIST VECTOR JUMP AND
                ;THEN BRANCH TO IT

                PRNCHR:

011A 2A0100             LHLD    WBOOT   ;LOAD H,L WITH WARM BOOT ADDRESS
011D 110C00             LXI     D,LIST  ;LOAD D,E WITH OFFSET
0120 19                 DAD     D       ;COMPUTE ADDRESS
0121 E9                 PCHL            ;BRANCH

0122                    END
```

```
                ;**************
                ;* LISTING 2 *
                ;**************

                ;THIS IS A DEMONSTRATION OF A VERY SIMPLE
                ;INTERRUPT DRIVEN OUTPUT ROUTINE. THE MEMORY
                ;LOCATION LABELED INTSTAT IS USED INDICATE
                ;WHETHER OR NOT A CHARACTER HAS BEEN PRINTED
                ;DURING AN INTERRUPT.


0031 =          INTEN   EQU     31H     ;BYTE USED TO INITIALIZE INTERRUPTS
                                        ;ON I/O PORT FOR OUTPUT ONLY
0011 =          INTDIS  EQU     11H     ;DISABLES INTERRUPTS ON PORT
0000 =          CSTAT   EQU     00      ;STATUS PORT
0001 =          CDATA   EQU     01      ;DATA PORT


0008                    ORG     08H     ;LOACATION ENTERED AFTER RST 1

0008 C31401             JMP     INTSRV  ;BRANCH TO INTERRUPT SERVICE ROUTINE

0100                    ORG     100H    ;USED FOR DEMONSTRATION PURPOSES

                CONOT:
0100 3A2701             LDA     INTSTAT ;PUT INTERRUPT STATUS IN A
0103 A7                 ANA     A       ;SET FLAGS
0104 C20001             JNZ     CONOT   ;LOOP IF CHARACTER WAITING FOR
                                        ; INTERRUPT
0107 3C                 INR     A       ;SET A TO 1
0108 322701             STA     INTSTAT ;SAVE CHARACTER READY FOR INTERRUPT
010B 79                 MOV     A,C     ;GET CHARACTER
010C 322801             STA     OUTCHR  ;SAVE CHARACTER
010F 3E31               MVI     A,INTEN ;ENABLE INTERRUPTS AT PORT
0111 D300               OUT     CSTAT   ;INITIALIZE PORT
0113 C9                 RET

                ;OUTPUT INTERRUPT SERVICE ROUTINE
                ;REGISTERS USED ARE SAVED TO PREVENT LOSS
                ;OF DATA NEEDED BY MAIN PROGRAM
                ;NOTE: THIS ROUTINE WOULD BE ENTERED VIA A JUMP
                ;FROM ONE OF THE 8 LOCATIONS BETWEEN 00 AND 38H.
                ;THE LOCATION IS DETERMINED BY THE DATA PLACED
                ;ON THE BUS BY THE I/O PORT HARDWARE DURING THE
                ;INTERRUPT.

                INTSRV:

0114 F5                 PUSH    PSW     ;SAVE A AND STATUS FLAGS
0115 C5                 PUSH    B       ;SAVE B,C REGISTER PAIR
0116 3A2801             LDA     OUTCHR  ;GET CHARACTER
0119 D301               OUT     CDATA   ;PRINT IT
011B AF                 XRA     A       ;SET A TO 0
011C 322701             STA     INTSTAT ;RESET INTERRUPT STATUS
011F 3E11               MVI     A,INTDIS;GET INITIALIZATION BYTE
0121 D300               OUT     CSTAT   ;INITIALIZE PORT
0123 C1                 POP     B       ;GET DATA BACK
0124 F1                 POP     PSW     ;GET DATA BACK
0125 FB                 EI              ;ENABLE INTERRUPTS
0126 C9                 RET             ;RETURN TO MAIN PROGRAM


0127 00         INTSTAT:DB      00      ;INTERRUPT STATUS
0128 00         OUTCHR: DB      00      ;CHARACTER STORAGE
```

```
                ;**************
                ;* LISTING 3 *
                ;**************

                ;THIS LISTING DEMONSTRATES INTERRUPT DRIVEN INPUT AND
                ;OUTPUT. THESE ROUTINES WERE EXTRACTED FROM A WORKING
                ;BIOS AND ARE ONLY MINIMALY COMMENTED. THE INPUT SERVICE
                ;ROUTINE ALLOWS THE USER TO EMULATE CP/M CNTR-P OPTION -
                ;PRINTING ON LIST DEVICE OR PROGRAM TERMINATION - CP/M
                ;CNTR-C OPTION.

                ;THE SERIAL BOARD USED IS THE MITS 2SIO WHICH USES THE
                ;MOTOROLA 6850 ASYNCHRONOUS COMMUNICATIONS INTERFACE.
                ;THIS CHIP IS INTIALIZED BY OUTPUTING 0B1H FOR INPUT AND
                ;OUTPUT INTERRUPTS, 031H FOR OUTPUT ALONE, 91H FOR INPUT
                ;ALONE, AND 11H FOR NO INTERRUPTS (POLLED I/O MODE)


                EQU     BUFLEN  10H     ;DETERMINES SIZE OF INPUT BUFFER

                ;
                ;INTERUPT SERVICE ROUTINE
                ;CHECKES CSTAT TO FIND WHICH PORT, IN OR OUT
                ;INTERUPTED. IF BOTH AT SAME TIME WILL SERVICE
                ;IN FIRST

                SRVINT: PUSH    PSW
                        IN      CSTAT   ;GET CONSOLE STATUS
                        ANI     01      ;CHECK FOR INPUT READY
                        JNZ     INTSRV  ;IF SO GO FOR INPUT
                        JMP     OINTSRV ;OTHERWISE GO FOR OUTPUT

                ; CHECK CONSOLE INPUT STATUS.
                ;
                CONST:  LDA     COUNT   ;GET NUMBER OF CHARACTERS WAITING
                                        ; TO BE INPUT
                        ANA     A       ;SET FLAGS
                CONST1: MVI     A,00    ;SET A = 0
                        RZ              ;RETURN IF COUNT=0
                        CMA             ;COMPLEMENT A, A=0FFH
                        RET
```

```
;
; READ A CHARACTER FROM CONSOLE.
;
CONIN:
        LDA     COUNT
        ANA     A
        JZ      CONIN   ;LOOP TILL INTERRUP
        DCR     A
        STA     COUNT
        LHLD    POINT2  ;PUT POINTER 2 IN H,L
        MOV     B,M     ;GET CHARACTER
        LDA     POS2
        DCR     A
        JZ      CONIN2
        INX     H
CONIN1: STA     POS2
        SHLD    POINT2
        EI
        MOV     A,B
        ANI     7FH
        RET
;
;THIS ROUTINE SET POINTER AND POSITION TO
;BEGINNING OF BUFFER
;

CONIN2: MVI     A,BUFLEN
        LXI     H,BUFFER
        JMP     CONIN1



;INTERUPT SERVICE ROUTINE
;

INTSRV: PUSH    H
        IN      CDATA   ;GET DATA
        ANI     7FH     ;STRIP PARITY BIT
        CPI     1BH     ;ESC = ESCAPE
        JZ      ATTN
        LHLD    POINT1  ;GET POINTER
        MOV     M,A     ;SAVE CHARACTER
        LDA     COUNT   ;GET CHARACTERS
        INR     A       ;UPDATE COUNT
        STA     COUNT
        LDA     POS1    ;GET BUFFER POSITION
        DCR     A       ;DECREMENT POSITION
        JZ      INTSRV2
        INX     H       ;UPDATE POINTER
INTSRV1:STA     POS1
        SHLD    POINT1  ;SAVE IT
INTSRV3:POP     H
        POP     PSW
        EI
        RET

INTSRV2:MVI     A,BUFLEN
        LXI     H,BUFFER
        JMP     INTSRV1

;ROUTINE TO CHECK FOR SPECIAL CONTROL CHARACTERS

ATTN:   IN      CSTAT
        ANI     01
        JZ      ATTN
        IN      CDATA
        ANI     7FH
        CPI     'C'     ;WARM BOOT
        JZ      00      ;TERMINATE
        CPI     'P'     ;PRINT TOGGLE
        JNZ     INTSRV3
        LDA     PRNECO
        XRI     1       ;TOGGLE BIT 0
        STA     PRNECO
        JMP     INTSRV3

; WRITE A CHARICTER TO THE CONSOLE DEVICE.
;THIS IS EXPERIMENTAL OUTPUT INTERUPTED ROUTINE
;
CONOT:
        LDA     ISTAT
        ANA     A
        JNZ     CONOT
        INR     A
        STA     ISTAT
        MOV     A,C
        STA     OUTCHR
        MVI     A,0D1H
        OUT     CSTAT
        RET

OINTSRV:

        PUSH    B
        LDA     OUTCHR
        OUT     CDATA
        MOV     C,A
        XRA     A
        STA     ISTAT
        MVI     A,021H
        OUT     CSTAT
        LDA     PRNECO  ;GET PRINTER STATUS
        ANI     01
        CNZ     LIST    ;IF PRINTER IS ON, PRINT CHARACTER
        POP     B
        POP     PSW
        EI
        RET

ISTAT:  DB      00
OUTCHR: DB      00
PRNECO: DB      00      ;1=PRINTER ON, 0=OFF

COUNT   DB      00
POINT1  DW      BUFFER
POS1    DB      BUFLEN
POINT2  DW      BUFFER
POS2    DB      BUFLEN
BUFFER  DS      BUFLEN



;**************
;* LISTING 4 *
;**************

;THIS LISTING SHOWS AN EXAMPLE OF IMPLEMENTING
;CHARACTER DELETION USING BACKSPACE FOR CP/M 1.4
```

```
                ;AND EARLIER SYSTEMS. THE MEMORY LOCATION LABELED
                ;DELSTAT, DELETION STATUS, IS USED BY THE OUTPUT
                ;ROUTINE TO DETERMINE WHEN TO ERASE A CHARACTER
                ;FROM THE CONSOLE SCREEN. THIS PROCEDURE SHOULD ONLY BE
                ;USED WITH CRT DEVICES AS OPPOSED TO TTY'S.


0010 =          CSTAT   EQU     10H
0011 =          CDATA   EQU     CSTAT+1


0100                    ORG     100H    ;USED FOR DEMONSTATION PURPOSE
0100 DB10       CONIN:  IN      CSTAT   ;GET STATUS
0102 E601               ANI     01      ;CHECK BIT 0
0104 CA0001             JZ      CONIN   ;LOOP TILL READY
0107 DB11               IN      CDATA   ;GET CHARACTER
0109 E67F               ANI     07FH    ;RESET PARITY BIT TO 0
010B FE7F               CPI     7FH     ;IS IT DELETE/RUBOUT
010D C0                 RNZ             ;RETURN IF NOT SO
010E 3EFF               MVI     A,0FFH  ;GET DELETION IN PROGRESS BYTE
0110 323A01             STA     DELSTAT ;SAVE IT
0113 C9                 RET

                CONOT:
0114 DB10               IN      CSTAT   ;GET PORT STATUS
0116 E602               ANI     02      ;SET FLAG
0118 CA1401             JZ      CONOT   ;LOOP TILL PORT READY
011B 3A3A01             LDA     DELSTAT ;GET DELETION STATUS
011E A7                 ANA     A       ;SET FLAGS
011F C22601             JNZ     CONOT1  ;JUMP IF NOT ZERO
0122 79                 MOV     A,C     ;GET CHARACTER
0123 D311               OUT     CDATA   ;PRINT IT
0125 C9                 RET

0126 AF         CONOT1: XRA     A       ;CLEAR A
0127 323A01             STA     DELSTAT ;CLEAR DELETION STATUS
012A 0E08               MVI     C,08    ;GET BACKSPACE
012C CD1401             CALL    CONOT   ;PRINT IT
012F 0E20               MVI     C,20H   ;GET SPACE - WILL CLEAR DELETED
                                        ; CHARACTER FROM SCREEN
0131 CD1401             CALL    CONOT   ;PRINT IT
0134 0E08               MVI     C,08    ;GET BACKSPACE
0136 CD1401             CALL    CONOT   ;PRINT IT
0139 C9                 RET

013A 00         DELSTAT:DB      00      ;FFH=DELETION,00=NO DELETION
013B C9                 RET


                ;**************
                ;* LISTING 5 *
                ;**************

                ;THIS SKELETON LISTING IS A DEMONSTRATION OF HOW TO
                ;PROGRAM A BIOS WITH IOBYTE IMPLEMENTED.
                ;ALTHOUGH THE ACTUAL DRIVERS ARE NOT LISTED, EACH ROUTINE IS
                ;SUPPLIED SO THAT THE READER CAN GET A FEEL FOR HOW THE SYSTE
                ;WORKS. THE VARIOUS DEVICES IN A HYPOTHETICAL SYSTEM COULD BE
                ;
                ;    TTY - AN ASR TYPE TTY
                ;    CRT - 9600 BAUD TERMINAL WITH DETACHABLE KEYBOARD
                ;    BAT - A MODEM BOARD
                ;    UC1 - INPUT - FROM CRT DETACHABLE KEYBOARD
                ;          OUTPUT - VIDEO BOARD FOR HIGH SPEED WORDPROCESSING
                ;    LPT - 180 CPS DOT MATRIX PRINTER
                ;    UL1 - 45 CPS DAISYWHEEL PRINTER FOR WORDPROCESSING

                ;THE LISTING ALSO SHOW HOW TO COMPUTE THE STARTING POINT
                ;OF BIOS GIVEN VERSION AND MEMORY SIZE.

0003 =          IOBYTE  EQU     03      ;IOBYTE IS LOCATED AT LOCATION 3
0014 =          MSIZE   EQU     20      ;USER WOULD FILL THIS IN FOR SYSTEM
1600 =          SYSTEM  EQU     800H+0E00H;LENTH OF 2.0 CCP AND BDOS
0000 =          BIAS    EQU     MSIZE-20;SET UP FOR CP/M 2.0 (SEE ARTICLE
3400 =          TPA     EQU     3400H   ;TPA SIZE IN MINIMAL SYSTEM + 100H

4A00                    ORG     TPA+SYSTEM+(BIAS*400H)


                ;VECTOR JUMP TABLE FOR A 20K SYSTEM

4A00 C3124A             JMP     BOOT    ;COLD BOOT
4A03 C3134A             JMP     WBOOT   ;WARM BOOT
4A06 C3144A             JMP     CONST   ;CONSOLE INPUT STATUS
4A09 C3294A             JMP     CONIN   ;CONSOLE INPUT
4A0C C33F4A             JMP     CONOT   ;CONSOLE OUT
4A0F C3534A             JMP     LIST    ;LIST DEVICE

                ;TABLE CONTINUES ON FROM HERE


                ;COLD BOOT ROUTINE WOULD GO HERE IN ACTUAL LISTING.
                ;RET USED SO THAT ASSEMBLER WILL WORK WITHOUT ERRORS.

4A12 C9         BOOT:   RET             ;FOR DEMO

                ;WARM BOOT ROUTINE WOULD GO HERE IN ACTUAL LISTING

4A13 C9         WBOOT:  RET             ;FOR DEMO

                ;CONST IS USED TO DETERMINE IF A CHARACTER IS READY
                ;AT THE CONSOLE DEVICE. WHICH CONSOLE DEVICE THAT IS TESTED
                ;IS DETERMINED BY I/O BYTE.

                CONST:
4A14 3A0300             LDA     IOBYTE  ;GET IOBYTE
4A17 E603               ANI     03      ;CHECK CONSOLE SEGMENT
                                        ;  AND STRIP OTHER SEGMENTS
4A19 CA6B4A             JZ      TTYST   ;CHECK TTY INPUT STATUS
4A1C FE01               CPI     01      ;SET FLAGS
4A1E CA6B4A             JZ      CRTST   ;CHECK CRT INPUT STATUS
4A21 FE02               CPI     02      ;SET FLAGS
4A23 CA6E4A             JZ      BATST   ;CHECK BATCH INPUT STATUS
4A26 C36B4A             JMP     CRTST   ;CHECK USER 1 INPUT STATUS

                ;ROUTINE TO DIRECT PROGRAM TO INPUT ROUTINE
                ;SPECIFIED BY IOBYTE

                CONIN:
```

```
4A29 3A0300          LDA     IOBYTE  ;GET IOBYTE
4A2C E603            ANI     03      ;CHECK CONSOLE SEGMENT
                                     ;  AND STRIP OTHER SEGMENTS
4A2E CA694A          JZ      TTYIN   ;INPUT FROM TTY
4A31 FE01            CPI     01      ;SET FLAGS
4A33 CA6C4A          JZ      CRTIN   ;INPUT FROM CRT
4A36 FE02            CPI     02      ;SET FLAGS
4A38 CA6F4A          JZ      BATIN   ;INPUT FROM BATCH DEVICE
4A3B C36C4A          JMP     CRTIN   ;INPUT FOR USER 1 DEVICE


                ;ROUTINE TO DIRECT PROGRAM TO CONSOLE OUTPUT ROUTINE
                ;AS SPECIFIED BY IOBYTE.

                CONOT:

4A3E 3A0300          LDA     IOBYTE  ;GET IOBYTE
4A41 E603            ANI     03      ;CHECK CONSOLE SEGMENT
                                     ;  AND STRIP OFF OTHERS
4A43 CA6A4A          JZ      TTYOT   ;OUTPUT TO TTY
4A46 FE01            CPI     01      ;SET FLAGS
4A48 CA6D4A          JZ      CRTOT   ;OUTPUT TO CRT
4A4B FE02            CPI     02      ;SET FLAGS
4A4D CA704A          JZ      BATOT   ;OUTPUT TO BATCH DEVICE
4A50 C3714A          JMP     UC1OT   ;OUTPUT TO USER 1 DEVICE

                ;ROUTINE TO DIRECT PROGRAM TO LIST DRIVER AS SPECIFIED
                ;BY IOBYTE

                LIST:

4A53 3A0300          LDA     IOBYTE  ;GET IOBYTE
4A56 E6C0            ANI     0C0H    ;CHECK LIST SEGMENT
                                     ;  AND STRIP OFF OTHERS
4A58 CA6A4A          JZ      TTYOT   ;OUTPUT TO TTY
4A5B FE40            CPI     40H     ;SET FLAGS
4A5D CA6D4A          JZ      CRTOT   ;OUTPUT TO TTY
4A60 FE80            CPI     80H     ;SET FLAGS
4A62 CA724A          JZ      LPTOT   ;OUTPUT TO LINE PRINTER
4A65 C3734A          JMP     UL1OT   ;OUTPUT TO USER LIST DEVICE

                ;ACTUAL DEVICE DRIVERS WOULD BE PLACED HERE, FOR DEMONSTRATIO
                ;PURPOSES, RET INSTRUCTIONS ARE USED.

                ;**** TTY ****

                ;ROUTINE TO CHECK INPUT STATUS OF TTY

                TTYST:

4A68 C9              RET     ;FOR DEMO

                ;ROUTINE TO INPUT FROM TTY

                TTYIN:

4A69 C9              RET     ;FOR DEMO


                ;ROUTINE TO PRINT TO TTY

                TTYOT:

4A6A C9              RET     ;FOR DEMO

4A6B C9              RET     ;FOR DEMO

                ;**** CRT ROUTINES ****

                ;ROUTINE TO CHECK CRT INPUT STATUS

                CRTST:

                RET     ;FOR DEMO

                ;ROUTINE TO INPUT FROM CRT

                CRTIN:

4A6C C9              RET     ;FOR DEMO

                ;ROUTINE TO PRINT TO CRT

                CRTOT:

4A6D C9              RET     ;FOR DEMO
                ;**** BATCH ROUTINES ****

                ;ROUTINE TO CHECK MODEM INPUT STATUS

                BATST:

4A6E C9              RET     ;FOR DEMO

                ;ROUTINE TO INPUT FROM MODEM

                BATIN:

4A6F C9              RET     ;FOR DEMO
                ;ROUTINE TO PRINT TO MODEM

                BATOT:

4A70 C9              RET     ;FOR DEMO

                ;*** USER CONSOLE 1 ****

                ;PRINTOUT TO VIDEO CARD

                UC1OT:

4A71 C9              RET     ;FOR DEMO

                ;**** LIST DEVICES ****

                ;OUT PUT TO A DOT MATRIX PRINTER

                LPTOT:

4A72 C9              RET     ;FOR DEMO
                ;OUTPUT TO DAISYWHEEL PRINTER

                UL1OT:

4A73 C9              RET     ;FOR DEMO

4A74                 END
```

Chapter II
The CP/M Connection

The CP/M Connection

Chris Terry

Interfacing to the Operating System: Relocating CP/M — Part 1

The CP/M system requires at least 16K of contiguous RAM for a minimal system. Page 0 is always reserved for entry points, file control blocks, and a 128-byte buffer used for command input from the console and as the default disk input/output buffer. Other buffer locations can be specified by an application program with a function call to BDOS.

The CP/M system proper is always located at the top of the available memory; in the minimal 16K system distributed by most disk controller manufacturers, the CCP starts at 2900H and the CBIOS at 3E00H. When more memory becomes available, the system can be relocated to the top of the new memory, so as to leave more room in the Transient Program Area (TPA) for application programs and data. This feature makes CP/M extremely versatile, because the addressable memory area above the RAM block can be used for PROM containing I/O routines and utilities without conflicting in any way with the CP/M requirements.

Four CP/M utilities are required for creating a relocated system and putting it on a fresh diskette:

MOVCPM
ASM
DDT
SYSGEN

Obviously, moving the CCP, BDOS, and CBIOS to a new location requires that all of the CALL and JMP addresses be changed to fall within the new system area. Equally obviously, we cannot change anything in the system that is currently up and running or it would crash. Therefore, to relocate the system, we use the MOVCPM utility, which contains a complete set of the system machine code. If we wish to create a new 32K CP/M system, we invoke MOVCPM with the command:

A>MOVCPM 32 *

MOVCPM now changes all the CALL and JMP addresses in its internal version of CP/M to suit the size we have requested (in this case, 32K), and then places the reconstructed CP/M code in the TPA with the Boot (Cold Start Loader) starting at 900H, the CCP starting at 980H, and the BIOS starting at 1E80H. Now it tells us that the new system is ready for SYSGEN or for the command:

SAVE 32 CPM32.COM

and DDT will automatically put the reconstructed CP/M at the right place (980H).

Let's look at figure 1. We see that our Boot has to be loaded at 900H. The execution addresses in BOOT.HEX start at 0000, and if we just used the simple Read (R) command of DDT, that is where the Boot would be loaded. However, DDT allows us to use an OFFSET with the read (R) command; this offset is added to every load address. We calculate it by taking the difference between the address where we want loading to start and the ORG address of the file. If BOOT is ORGed at 0000, the offset is 0900H-0000=900H; if BOOT is ORGed at 80H, the offset is 0900-0080H=880H. If we don't have a hex calculator (such as the TI Programmer), we can use the hex arithmetic command (H) of DDT; the command:

-H900,80

will cause DDT to give us first the sum (980) and then difference (880) of our two numbers. So, to overlay the MDS Boot with our own, we give the commands:

-IBOOT.HEX
-R900 (if ORG is 0000)
or
-R880 (if ORG is 80H)

Things become a little more tricky when we come to the CBIOS. Look at figure 1 for a moment. Moving the Boot to the Memory Image area was simple, because we were moving it upward. But to get the CBIOS shifted from its execution address of 7E00 to 1E80 in the Memory Image area, we have to shift it DOWNWARD. Unfortunately, DDT can only ADD an offset to, not subtract it from, the file load address. However, we can still use a positive offset that will bring us to the right place, because the CPU address counter has only 16 bits; thus, if we add 1 to address FFFF we get 10000 -- but the counter has no place to put the leftmost digit, so we come back to 0000. We see, then, that to shift a program downward in memory, we have to give DDT an offset that will push the program up off the top of memory and bring it upward through the bottom. A Two's Complement subtraction of the larger address from the smaller will do precisely this.

We know that for a 16K system the offset is 980-2900=E080, and since 3E00 is the execution

Chris Terry, 324 E. 35th St., New York, NY 10016

address of CBIOS, it will be found in the Memory Image area at:

3E00 + E080 = 11E80 = 1E80

We also know that the start of the CCP in our new system is 2900+4000=6900, so the offset for our new system is 980-6900=A080. If we add this to the start of our new CBIOS, which is 3E00+BIAS=7e00, we find that 7E00+A080=1E80. Bingo! Now, to overlay the MDS BIOS (which was put in the Memory Image area at 1E80 by MOVCPM) we tell DDT:

```
-ICBIOS.HEX
-RA080
```

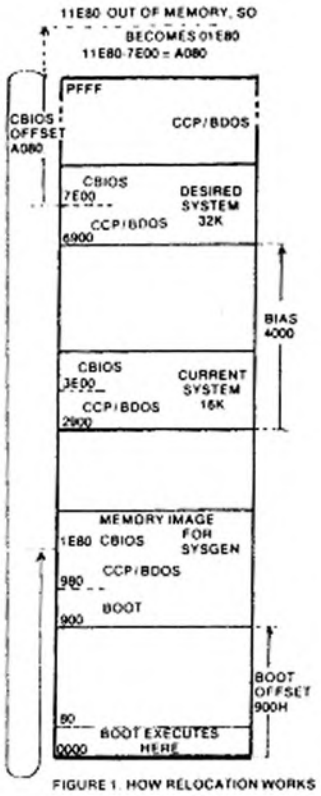


FIGURE 1. HOW RELOCATION WORKS

We can forget about SYSGEN at this point, because the Boot and the BIOS generated by MOVCPM are for the Intel MDS system and chances are that not even the console I/O would work, let alone the disk commands. So we save the COM file, as instructed.

Now we have some preparation work to do, so that we can overlay the MDS Boot and BIOS with the Boot and CBIOS supplied by our controller manufacturer and previously modified to work with our system I/O. First, we must calculate the BIAS for our new system; that is, the amount by which every CP/M instruction is shifted upward. MOVCPM took care of this for the CCP and BDOS instructions, but we need it to find where to ORG our own CBIOS. We are going to move the system up by 16K; since 4096 decimal (4K) is equivalent to 1000 hex, it follows that our BIAS for a 32K system is going to be 4x1000=4000H. We have to apply this BIAS to three items:

In the Coldstart Loader, to set:
(1) the address at which CP/M will be loaded, and
(2) the address to which the loader will jump to start CP/M running;
In the CBIOS, to set:
(3) the ORG address.

Let's look at the ASM listing of the Boot (using our editor), and find out where it executes. It will almost certainly be either 0000 (as in the case of the Tarbell disc controller board) or 80H (as in the case of the Thinker Toy board). We make a note of the ORG address, but do not change it. If we find an MSIZE EQU 16 statement, this will be used by the assembler to compute the load address (1) and the jump address (2); we need only change it to MSIZE EQU 32. If we do not find an MSIZE equate, the 4th executable statement will be LXI H,2900H; we change the operand to LXI H,6900 (=2900+BIAS). The 3rd statement of the RBLK1 routine will be JZ 3E00H; we change this to JZ 7E00H (=3E00+BIAS).

Next, we must go into the CBIOS.ASM file with our editor, and look at the ORG statement. Once again, if we find an MSIZE equate statement, that is all we need to change. The Assembler will do the rest.

If we do NOT have an MSIZE statement, but instead find ORG 3E00H (or any other absolute address), we have to change the ORG address to 3E00+BIAS (in this case 7E00). Assuming that we don't want to make any changes to the peripheral drivers (for console, list, reader, or punch), we can exit from the editor.

Now we use ASM to reassemble the CBIOS at the new address. It's a good idea to let the assembler create a PRN listing with all the addresses and object code, and to print this right away. We can then ERAse CBIOS.PRN, which takes up a lot of room on the disk and is not required any more. Before going any further, let's check the directory and make sure that we do indeed have the CPM32.COM, CBIOS.HEX, and BOOT.HEX files.

CALCULATING OFFSET

At this point we have to remember that every COM and .HEX file contains built-in instructions on where to load it. We shall have no trouble with the CPM32.COM which we saved previously, because MOVCPM arranged for the reconstructed CCP and BDOS to be loaded into the Memory Image area starting at 980H, even though they will execute at 2900+BIAS (6900 in this case). We have only to give the command:

A>DDT CPM32.COM

```
A>MOVCPM 32 *                          Reconstruct system

CONSTRUCTING 32K CP/M VERS 1.4
READY FOR "SYSGEN" OR                  CPM's reply
"SAVE 32 CPM32.COM"
A>SAVE 32 CPM32.COM                    Save the new system
A>ASM TTCBIOS                          Assemble edited CBIOS
CP/M ASSEMBLER - VER 1.4
3F4D
004H USE FACTOR
END OF ASSEMBLY

A>ASM TTBOOT                           Assemble edited Boot
CP/M ASSEMBLER - VER 1.4
0100
001H USE FACTOR
END OF ASSEMBLY

                                       Ask for directory
A>WDIR
-WORK    011     ASM      COM     COPY3    COM     CUTER    COM
DDT      COM     DIAPRINT HEX     DPRT12   COM     INTLIZE  ASM
IODVRS   ASM     MOVCPM   COM     NEWCBIOS ASM     NTARBIOS ASM
NTARBIOS ASM+1   NTARBIOS HEX     PAGE     COM     PIP      COM
SAP      COM     SBOOT40  ASM     SBOOT40  HEX     SBOOT40  PRN
STAT     COM     SUBMIT   COM     SYSGEN   ASM     SYSGEN   COM
SYSGEN   HEX     SYSGEN   PRN     SYSGEN   PRN+1   SYSGEN   SYM
TARBIOS4 ASM     TARBIOS4 ASM+1   TARBIOS4 HEX     TTBOOT   ASM
TTCBIOS  ASM     WDIR     COM     WM       COM     XFER     COM
CBIOS64  HEX     CPM64    COM     SBOOT64  HEX     PINIT    COM
CPM32    COM     TTCBIOS  PRN     TTCBIOS  HEX     TTBOOT   PRN
TTBOOT   HEX
A>
A>DDT CPM32.COM                        Get new system into Memory Image
DDT VERS 1.4                                area
NEXT  PC
2100 0100
-ITTBOOT.HEX                           Create FCB for Boot
-R880                                  Overlay MDS Boot with ours
NEXT  PC
2100 0000
-ITTCBIOS.HEX                          Create FCB for CBIOS
-RA080                                 Overlay MDS BIOS with ours
NEXT  PC
2100 0000
-^C                                    New system complete; reboot current system
A>SYSGEN
SYSGEN VER 1.403                       CPM's reply
FOR PERTEC SINGLE DENSITY DISK
SOURCE DRIVE NAME (OR RETURN TO SKIP)  Return, because we have
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B    Memory Image
DESTINATION ON B, THEN TYPE RETURN     Write new system to B
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)     Return, to reboot
                                                 current system
```

FIGURE 2. SAMPLE RELOCATION JOB

If it should be necessary at this stage to make minor changes to the CBIOS, it is now easy to find the address at which the change is to be made. Any address in the Memory Image area can be found by adding the offset to the execution address shown in the CBIOS.PRN listing of the re-assembled CBIOS. We can then use either the DDT Substitute (S) command to insert the new hex values or, if several successive instructions are to be changed, we can use the DDT A (Assemble) command which allows complete instructions to be inserted using the Intel mnemonics for operation and register codes, and hex values for addresses or constants.

The CP/M System Alteration Manual has all this information; there is even a table of offsets for various system sizes. But for some reason I and many others have great difficulty in getting the procedure clear. Perhaps the CP/M manual gives us too much, too quickly -- that is why I have spreadout this description.

THE USES OF 'SYSGEN'

At this point, the Memory Image area contains a complete 32K CP/M system. with the correct CBIOS and Boot for our own configuration. We now use the SYSGEN utility to write the new system out to Tracks 0 and 1 of a fresh, formatted disk. The complete sequence of commands is shown in figure 2, with comments.

Note that when SYSGEN asks SOURCE DRIVE NAME (OR RETURN TO SKIP), we hit Return because we already have the reconstructed system in the memory area. However, when we are not relocating CP/M, but merely putting the existing system, unchanged, onto a new disk, we do not need to use DDT to get the current system into the Memory Image area. When SYSGEN asks for the source drive, we tell it A, and CP/M is read from Tracks 0 and 1 of our current system disk into the Memory Image area. Then, when SYSGEN asks for the destination, we tell it B, and write the system out to a new disk.

COPYING THE SYSTEM FILES

SYSGEN does not handle anything except the CP/M system itself. The utilities, such as ASM, ED, DUMP, LOAD, etc., must be handled separately. The files on our current system disk can be transferred over to the new system either by the CP/M Users' Group utility COPY.COM, or by the command

```
A>PIP B:=A:*.*[V]
```

COPY transfers a whole track at a time, verifying each sector but not reporting until the end. PIP transfers one file at a time, verifies the new file against the old, and reports the name of the file transferred, so it takes somewhat longer. PIP will be our choice if we want to be selective, copying only the .COM files, for example.

The CP/M Connection

Chris Terry

CP/M File Operations — Part 2

CP/M is available for a variety of disk drives, controllers and methods, including single and double-density, hard and soft sectoring, 8-inch and 5-1/4-inch disks and Winchester hard disk drives, all of which vary considerably in their disk primitives. In this article, for the sake of simplicity, we consider only the standard distribution version of CP/M Version 1.4, issued on a single-density, soft-sectored, 8-inch disk.

DISK ORGANIZATION

Main Divisions of Disk Space

The standard soft-sectored, single-density, 8-inch disk is divided into 77 Tracks (numbered 0 through 76), and there are 26 Sectors (numbered 1 through 26) per track. This conforms to the IBM 3740 disk layout; such disks are called "IBM-compatible".

Each sector stores 128 data bytes; the two Cyclic Redundancy Check bytes and other overhead bytes which follow the data are not included in this count. Thus, the total storage space is 77*26*128=256,256 bytes. This, too, follows the IBM format, but again is a function of a BDOS table; it is perfectly possible to set the sector size to any multiple of 128 by changing the table entry, but files would not then be portable except to another system with the same blocking factor.

On every disk that runs under CP/M, the storage space is divided into three distinct areas:

- CP/M System Area
- File Directory Area
- File Storage Area

CP/M System Area. Tracks 0 and 1 are always reserved for the CP/M system, although the system need not be present on every disk. The coldstart loader is contained in Track 0 Sector 1; the CCP and BDOS occupy the rest of Track 0 as well as 17 sectors on Track 1; the remaining nine sectors (1152 bytes) of Track 1 are available for the CBIOS. The number of sectors actually used by the CBIOS depends on what drivers and features are included by the controller manufacturer.

File Directory Area. Sixteen sectors on Track 2 are always reserved for the file directory. Each directory entry is 32 bytes long; thus, there is room for (16*128)/32=64 entries in the standard system. Note, however, that sector allocation for the directory is controlled by a table in the BDOS; OEMs licensed by Digital Research Inc. to reconfigure the system can expand the number of directory entries to 255 by changing this table.

File Storage Area. The remaining ten sectors on Track 2 and all sectors on Tracks 3 through 76 are available for files.

Logical/Physical Sector Mapping

The standard logical record is one sector (128 bytes), and a file may occupy any number of sectors from zero up to the full capacity of the disk. Logically consecutive records are not physically contiguous on the disk. This is because the disk controller must process the CRC bytes after reading Logical Sector N, to verify that there were no read errors. Also, the BDOS has some housekeeping chores to perform. If Logical Record N+1 were in fact physically adjacent to Record N, it would probably pass under the read head before the chores were complete; the system would then have to wait until it came round on the next revolution of the disk, about 16 milliseconds later. This would make sequential reading unacceptably slow.

For this reason, logically consecutive records are mapped onto the disk with several physical sectors between each. The standard skew (sometimes called "interlace") for CP/M is six sectors, to be IBM-compatible, and is shown in Figure 1; this mapping is identical for all directory and file storage tracks. The translation from logical record numbers to physical sector numbers is performed by a lookup table that is usually in BDOS, though some versions put the table in the BIOS. When the table is in the BDOS, disk utilities that use the disk primitives directly must provide a separate translation table of their own. Thus, an application program using the disk primitives to gain access to logical Sectors 19, 20, and 21 of Track 3, would in fact access physical sectors 6, 12, and 18 on that Track. After completing the housekeeping for Sector 6, there is only a minimal wait before Sector 12 (the next in the logical sequence) arrives under the read head.

The routine that performs the logical/physical sector mapping is transparent to the user. In effect this routine says, "Whenever you give me a logical record number, I will convert it to a physical sector number. You don't need to know what that number is, but my mapping will give you quicker access to the data area."

KEEPING TRACK OF DISK SPACE USAGE

Unlike some other microcomputer operating systems, CP/M does not require that the size of a file be specified at the time of creation. Instead, space on the disk is allocated dynamically, as needed. Space that is released as the result of closing a file from which at least 1K has been deleted, can immediately be re-used by another file. The tools that permit this dynamic space allocation are:

- The Allocation Bit Map
- The File Control Block (FCB)
- The Directory

Allocation Bit Map

For every drive configured in the system, the BDOS maintains a space allocation bit map consisting of 243 individual bits. This map is read into memory when the drive is logged in, is modified during Write operations, and is written back to the disk each time a file on that disk is closed. Typing Ctrl-C erases all bit maps from memory except those for Drive A and for the currently logged-in disk.

Each bit in the map represents a group (sometimes called a "cluster") of eight logically consecutive sectors on the disk. The bit positions and their associated groups are numbered 00 through F2 hex (see Figure 2). The first two bits are associated with the first sixteen logical sectors on Track 2. These two groups (00 and 01) contain the file directory, and bits 00 and 01 in the allocation map always contain 1's, even when no directory entries have yet been made. This ensures that the directory can never be overwritten by a file.

```
A>DISKDUMP B:MAP

GROUP ALLOCATION MAP DRIVE - B

                    Columns
                 0123456789ABCDEF0123456789ABCDEF
Even Half-rows   0 1111111111111111 1111111111111111  1  Odd Half-rows
                 2 1111111111111111 1111111111111111  3
                 4 1111111111111111 1111111111111111  5
                 6 1111111111111111 1100000000000000  7
                 8 0000000000000000 0000000000000000  9
                 A 0000000000000000 0000000000000000  B
                 C 0000000000000000 0000000000000000  D
                 E 0000000000000000 00                F

127 GROUPS REMAINING ON DISK OUT OF 243

To find the hex Group number of a bit in the map, take the first digit from the
even or odd half-row and the second digit from the column in the same half.
The first 0 represents 73.
```

Figure 2. Allocation Bit Map

When BDOS receives a request to create a file, it searches the allocation bit map until it finds a bit containing a 0; the number of this bit is the number of the first free group. BDOS then sets the map bit to 1 and places the 1-byte hexadecimal group number in the mapping area of the File Control Block (FCB) created for the new file.

Each time a Write operation is requested for the file, BDOS examines the last group number in the FCB and also the Next Record number, and from these computes the Track and Logical Sector numbers where writing is to take place. When all eight sectors of a group have been filled, BDOS automatically searches the allocation bit map again for the first bit containing a 0. When one is found, its group number is added to the FCB and the map bit is set to 1. Thus, a file which has seven or fewer records will be shown as occupying one group (1K); a file which has eight records will be shown as occupying two groups, even though the second group is empty.

SECTOR ID NUMBERS FORMATTED ON DISK	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
LOGICAL SECTOR #s																										
Read on Pass 1	01					02					03					04					05					
Read on Pass 2			06				07				08				09				10				11			
Read on Pass 3		12					13				14				15				16				17			
Read on Pass 4	18					19					20				21				22				23			
Read on Pass 5					24					25					26											
Read on Pass 6				01					02					03					04					05		

Figure 1. Standard 6-Sector Skew in Logical/Physical Sector Mapping.

This process has several important results:

- The minimum space that can be occupied by any file (even an empty file) is eight sectors (1K).
- Because BDOS always searches the allocation bit map from the beginning on a Write request and allocates the first free group it finds, logically consecutive components of a file may be physically located anywhere on the disk and not necessarily in Track/Sector order.
- A Write request will never be denied until the disk is too full to hold the amount of data to be written. Of course, denial of a Write request is a fatal error unless the application program makes provision for mounting a fresh disk in such circumstances, but it very seldom occurs if reasonable care is taken. Use the STAT utility to check available space before undertaking any operation that creates backup or temporary files.
- Disk space is efficiently used. In other systems that require file size to be specified, overcaution can result in large amounts of unused space that is not available to other files. This can only be recovered by copying the data to a new file with the proper size specification. The same procedure has to be followed if it is desired to expand a file that has already used the space originally allocated to it.

File Control Block (FCB)

An FCB is a 33-byte block of read/write memory containing all the information needed by BDOS to find a file on the disk and to access any specified record. Whenever a new file is created, an FCB must be created for it. The area from 005C to 007C hex is the default FCB area used by the CCP; it may also be used by transient programs. If a transient program requires more than one file to be open at the same time, the program must create an FCB for each file that is to be accessed. These FCBs should be in the TPA.

FCB Layout. The layout of an FCB is shown in Figure 3. When a file is first created, the CCP or user program first clears all bytes of the FCB to zero, and then initializes the first thirteen bytes as follows.

ET. Byte 0. The CP/M Manual defines this as "Entry Type, not currently used but assumed zero." While in the FCB area, this byte remains 0.

FN. Bytes 1 thru 8. The CCP or user program places the filename in this field, left-justified. If the name has fewer than eight characters, the remaining bytes are padded with ASCII blanks (20 hex).

FT. Bytes 9 thru 11. The CCP or user program places the 3-character file type in this field. Note that the period which separates filename and type in a command is only a delimiter and is not put in the FCB. If the file type has fewer than three characters, the remaining bytes of the FT field are padded with ASCII blanks. If the file is a temporary file, this field will contain '$$$'.

EX. Byte 12. This byte, initialized to zero, indicates the file extent number. As we shall see, an FCB describes a file segment up to 16K in size, i.e., 128 records (sectors). When 128 sectors have been written, bytes 0 thru 31 of the FCB are copied to the first free slot in the directory area, and the Extent number in the FCB is incremented. Thus, we shall find a separate directory entry, each containing a different extent number, for every 16K segment of a large file.

Bytes 13 and 14 are not used and should always be zero.

RC. Byte 15. This byte, initialized to 0, contains the current number of records in the Extent described by the FCB. As new records are written to the disk, this count is updated by BDOS. Transition of this count from 7F to 80 is the signal for BDOS to copy the FCB to the directory area of the disk and to create a new FCB with the EX and NR bytes updated.

DM. Bytes 16 thru 31. This is the Disk Map area, and is initialized to zeros. When a file is being built, the first Write request causes BDOS to insert the number of the group allocated into Byte 16. No further updating takes place in this field until all sectors of the group have been written. Then BDOS allocates another group and inserts its number into Byte 17, and so on, until all 16 groups (128 sectors) have been written.

IMPORTANT NOTE: Because the FCB is not written to the disk directory area until either 128 records have been written or the file is closed, a system crash can cause the apparent loss of up to 128 records. The data is on the disk but is not recorded in the directory. In applications that entail much data entry, it is good practice to close the file frequently and re-open it; this can avoid painful reconstruction of the directory and the possible destruction of vital data as the result of overwriting from other files after a crash.

NR. Byte 32. This byte, initialized to zero, is updated by BDOS during sequential file operations, and shows the number of the next record to be read or written. For random access, the transient program must place the number of the record to be accessed in this byte before issuing the function call to BDOS. Note that this byte is not copied to the directory entry; it is meaningful only when the file has been opened.

FCB Location. Before we go on to discuss the directory, it is important to emphasize that there is no restriction on the location of an FCB. The CCP and DDT use the area from 005C to 007C hex; this is known as the default FCB area, and is usually given the symbolic label TFCB. When a large file has more than one 16K extent, sequential write operations build the data for each extent in the default FCB area. Sequential read operations cause each directory entry for the file to be fetched into TFCB, in turn. However, a user program may allocate enough memory to hold all the FCBs of a file simultaneously, passing the address of the appropriate FCB to BDOS as one argument of each access request. This will be explained in more detail when we discuss file access operations. Much time and head movement can be saved during random read operations if all of the FCBs for a file are available in RAM, so that they do not have to be fetched from the disk each time a new extent is accessed.

File Directory

The BDOS maintains a directory for each disk. Upon booting CP/M, the contents of Groups 01 and 02

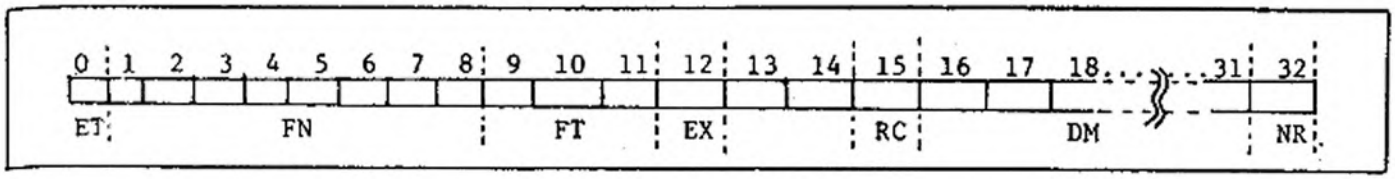


Figure 3. Layout of File Control Block

(16 sectors) are read sequentially from the logged-in disk, and the bit map for that drive is recalculated from the DM bytes of each entry. A request to open a file causes the appropriate directory entry to be copied into the FCB area. If the file has been opened for writing, closing the file causes BDOS to copy the FCB back to the directory area and to rewrite all 16 sectors to the disk immediately.

Directory Layout. A set of typical directory entries is shown in Figure 4. They are for the same disk as the Allocation Map in figure 3. Note that each entry occupies 32 bytes in two lines. The first line (3rd address digit always even) contains 00 in the first byte (to indicate that the entry is for a valid file), followed by the file name and type. The extent number is in byte nnnC, and the record count in nnnF. The second line (3rd address digit always odd) of the entry contains the numbers of the 8-sector groups allocated to the file.

Restoring Erased or Crashed Files. An understanding of this layout may help to recover a file after a system crash or one that has been accidentally erased. The CP/M ERAse command and the Basic KILL command do not in any way modify the disk file. They merely issue a Delete request to BDOS which places E5 in the first byte of the directory entry to mark it as deleted. BDOS also scans the group allocations in line 2 of the entry and sets the corresponding bits in the Allocation Map to 0, thereby freeing these groups for re-use. The file data on the disk remains intact until a Write request to BDOS finds one of these groups free and overwrites one or more sectors on the disk. The disk Dump utility by S.J. Singer on Volume 24 of the CP/M User's Group library can not only read any sector of the disk into an accessible area of memory, but allows examination and replacement of individual bytes before writing the sector back to disk. This facility can restore an erased file to activity by changing the E5 in the directory entry back to 00, provided that the directory entry is intact and that write requests since the erasure have not overwritten the file data on the disk. Restoration also requires Opening the file, to bring the edited directory entry into the FCB area, and then Closing it to restore the Allocation Map and to rewrite this and the restored directory back to the disk.

In the case where the system crashed while a file was open, this disk DUMP utility can also be used to search the file area of the disk for the lost data. If the first extent is preserved in the directory, exploration of the last groups used and the first groups ostensibly unused can provide the first clues on where to look. If two files are open simultaneously for writing, BDOS usually (though not always) allocates open groups alternately to these two files; this, too, can be a help in reconstructing directory entries.

The moral is, never, NEVER allow your application programs to keep writing and writing to a file. If you close and re-open the file every time you write a record, you will never lose any data. If this is too hard on the disk drive, close and re-open the file after every 4K has been written (or some other reasonable amount).

Directory Area Usage. It is worth noting that directory entries on the disk remain intact, even though marked as deleted, until they are overwritten by new entries. Thus, when the directory is brought into memory from the disk, it may contain entries marked for deletion which BDOS will re-use as needed. When BDOS copies an FCB to the directory area, it puts the FCB into the first entry slot that contains E5 in its first byte. As a result, entries for several extents of the same file are not necessarily adjacent, nor even in numerical order. The CP/M DIRectory command and the WDIR (Wide Directory) utility display file names in the order in which they occur in the directory.

If it is desired to alphabetize the directory and to purge entries for deleted files, the SAP (Sort and Pack) utility by Bruce Ratoff (CPMUG Volume 19) can be used. This utility reads the directory from the disk, copying only the active entries into an empty 2K buffer. It then sorts the selected entries into alpha-numeric filename-type-extent order, fills the rest of the buffer with E5, and then writes the sorted and purged directory back to the disk. BE CAREFUL, however. Early versions of SAP operate ONLY on Drive A and are constructed for a particular system size. And, to the best of my belief, existing versions of SAP do not work on double-density systems. Be sure to check the source code, and try it on a backup disk first.

User Program File Access Procedures

When a file-related console command is given (SAVE, ERA, REN, DIR, TYPE) with a drive, filename, and file type, or when a CP/M utility (STAT, PIP, ASM, DDT, etc.) is invoked with a filename as its argument, the CCP or the utility perform all functions required to access the named file, including the creation and updating of the FCB and directory entry. We are here concerned only with the procedures that must be performed by user programs to create, modify, or read data files. Some general principles are explained first; these are followed by some concrete examples.

BDOS Function Calls

CP/M provides 27 different functions, all of which are available to user programs. Functions 1 through 11 relate to peripheral I/O, and are discussed elsewhere. Functions 12 through 27 are disk I/O functions. Only those concerned with creating a new file, reading

from or writing to an existing or newly created file, or deleting a file are discussed here.

A BDOS function call (that is, a request to BDOS to perform some function) always consists of three operations:

- Load Register C with the function number of the desired operation.
- Load Register Pair DE with the address of the FCB for the file to be accessed. For function 26, load the address of the buffer to be used for disk reads and writes.
- CALL BDOS (entry point is 0005H).

Some, though not all, functions return a result. Single-byte results are returned in the A register. Double-byte results are returned with the low byte in the A register and the high byte in the B register. It is the responsibility of the user program to interpret and use any results returned by BDOS. NOTE: BDOS uses all the registers. If any register values have to be preserved, save them before the BDOS function call and restore them when the function is complete.

Log-In Disk (Function 14)

If the file to be accessed is not on the same disk as the user program, the drive on which the file is (or will be) stored must be logged in. That is, its Allocation Map must be reconstructed in memory before any access can be attempted. Put function number 14 (0EH) in the C register, clear the D register, and load E with the drive number to be logged in. Then call BDOS. No results are returned.

If your program calls for a change of disk, it MUST call for a log-in; if it does not, BDOS will attempt to use the allocation bit map left over from the previous disk on that drive, and existing data on the new disk may be overwritten and permanently lost.

Create a New File

First, allocate space for an FCB. If no other file is open, the TFCB at 005CH may be used; if that is already in use, allocate FCB space (33 bytes) in the transient program area (TPA). Then move the filename (8 characters, left-justified, padded as necessary with ASCII blanks) and the file type (3 characters, left-justified, padded if necessary) into bytes 1 through 11 of the FCB (byte 0 must contain zero).

Load Register C with function number 22 (16H, Make File), load Register Pair DE with the address of the FCB, and call BDOS. BDOS returns the byte address of the directory entry allocated to the file (i.e., an address in the range 00 through 7FH that is relative to the start of the sector in which the entry will be stored on the disk). If the directory is already full, BDOS returns 0FFH in the A register; the user program must check for this and take appropriate action if the directory is full. One possible course would be to print an error message instructing the operator to dismount the current disk and mount a blank formatted disk on the same drive. Upon receiving confirmation via the console that a new disk is mounted, start the operation over by logging in the disk (Function 14) and repeating the Make.

Opening an Existing File

If the file to be accessed already exists, it must be opened before reading or writing can take place. If the file is not on the same disk as the user program, it must be mounted and logged in as described above; if it is on the same disk, the log-in was done when the user program was called.

To open the file, do the following:

- Allocate space for the FCB.
- Move the filename and type into bytes 1 through 11 of the FCB as described for a Make, above.
- Load the C register with function number 15 (0FH, Open File). Load Register Pair DE with the with the address of the FCB.
- CALL BDOS (Entry point is 0005H).
- Clear register A and wait for the completion code to be returned. BDOS returns the byte address of the directory entry if the file is successfully opened, or 0FFH if the file cannot be found.

NOTE: Successful opening of a file says nothing about the mode in which it can be accessed. It merely indicates that the file exists and that its directory entry has been copied into the FCB area. The user program may either read from or write to the file, sequentially or randomly. If only Read operations are performed, the file need not be closed later (although it is good practice to do so). If any kind of Write operation is performed, the file MUST be closed later, in order to ensure that the added or modified space allocations are permanently recorded on the disk, both in the allocation bit map and in the directory.

Buffer Addressing

The starting address of the buffer from which data is to be written to disk, or into which data is to be read from disk, is called the DMA (Direct Memory Access) address. The minimum size of the buffer is 128 bytes, corresponding to one complete sector. Increases of buffer size must be in multiples of 128. The term "DMA" is not strictly accurate unless the controller contains DMA hardware that pre-empts the data bus and transfers a specified number of bytes (starting at the DMA address) at high speed, without intervention of the CPU. However, the term is convenient and has become standard in CP/M.

Unless otherwise specified by a user program, BDOS assumes that all data transfers will take place via the 128-byte buffer at locations 0080H through 00FFH. This is called the Default Buffer, and the standard name of its starting address is TBUFF. This buffer is also used by the CCP for string input from and output to the console.

A user program can change the disk buffer address with a Set DMA function call to BDOS. The procedure is:

- Load Register C with the function code (26=1AH=Set DMA).

- Load the DE Register pair with the starting address of a user buffer.
- CALL BDOS (Entry point at 0005H).

BDOS does not return anything.

Reading and Writing

After a file has been opened, a separate Read or Write request must be issued for each and every sector transferred. Unless otherwise specified by the user program, BDOS assumes that all transfers will take place via the Default Buffer starting at TBUFF (0080H). The user program is responsible for emptying the buffer after each Read (or processing the data while it is still in the buffer), and for filling the buffer before each Write. The request procedure is:

- Load the C Register with the function code (20=14H=Read Next Sector;21=15H=Write Next Sector.
- Load Register Pair DE with the address of the FCB for the file to be accessed.
- CALL BDOS (entry point is 0005H).

Upon completion of the sector Read or Write, BDOS increments the count in the NR (Next Record) field of the FCB, and returns a completion code in Register A. The completion codes are:

CODE	ON READ	ON WRITE
0	Successful Read	Successful Write
1	Read past EOF	Error in extending the file
2	Sector accessed	End of disk data had no data
255		No more directory space for a new extent

It is the user program's responsibility to check the completion code and to take appropriate action on error conditions. Use of the 128-byte Default Buffer is convenient when the user program must process small quantities of data (for example, a line of source code) before requesting or outputting another record.

There are many occasions when a large block of data must be read into or written from memory in one operation. Examples are reading a .COM file into memory prior to execution, reading a complete set of records that are to be sorted, or making a large block of ASCII text available to speed up string search/replace procedures. To read a large block, do the following:

- Allocate user buffer space, sized to some multiple of 128 bytes.
- Issue a Set DMA request pointing to the start of the user buffer. Store the current DMA address in scratchpad memory.
- Issue the first Read request.
- After each Read request, check the completion code returned by BDOS, and take appropriate action if an error is indicated. Also check for a Buffer Full condition (such as number of sectors read equal to buffer size in sectors).
- After each successful read, get the current DMA address, add 128 to it, and store the updated address in the DE register pair and in the scratchpad. Then issue a new Set DMA request followed by a Read request.

Random Reading

Once a file has been opened, random access to any record is possible by placing the desired record number in byte 31 of the FCB before issuing a Read or Write request (remember that FCB bytes are numbered from 0). Some computation is required here, first to derive the logical record number from the block number (if blocks are larger than 128 bytes), and then to find what extent this record is in.

Suppose that our logical records are 256 bytes (2 sectors) long, and we wish to access record 134. The data we want is in the two sectors starting at 134*2=268 in the sector sequence. However, since a file extent can hold only 128 sectors, sector 268 must be in the third extent. Extent numbers start at 0, so this will be numbered 02. The number of sector 268 relative to the start of Extent 02 is found by taking the remainder of 268 modulo 128; that is, 268%2=2, and the remainder is 268-(128*2)=12 decimal. Since Next Record counts in the FCB also start at 0, the required sector number is 11 (0BH).

Thus, before issuing the access request we must fetch the directory entry for Extent 02 into the FCB area, and place 0BH in the NR field (Byte 31). Now we issue a Set DMA request pointing to the start of a 256-byte buffer, followed by a Read request for the first half of our record. To obtain the second half, we must add 128 to the DMA address and issue a new Set DMA request. We do not need to change the NR field of the FCB because this was incremented automatically by BDOS after the first read, so we finish the operation merely by issuing the second Read request.

Random Writing

Some care must be taken when writing randomly. If we wish to write record 129, for example, we must first have created space for records 1 through 128. We can write 128 records containing nuls, and then add record 129 to the end of these; however, this may be wasteful of disk space, and we could run into trouble if we attempt to write record 2001 (or some high number). Most data management systems use a special CREATE program to create a file of finite size, and then an UPDATE program that enters data into this file in a manner that makes efficient use of the space. There have been a number of articles during the last year on hashing techniques, tree techniques, and indexed sequential access methods. Consult these for further details, which are outside the scope of this article.

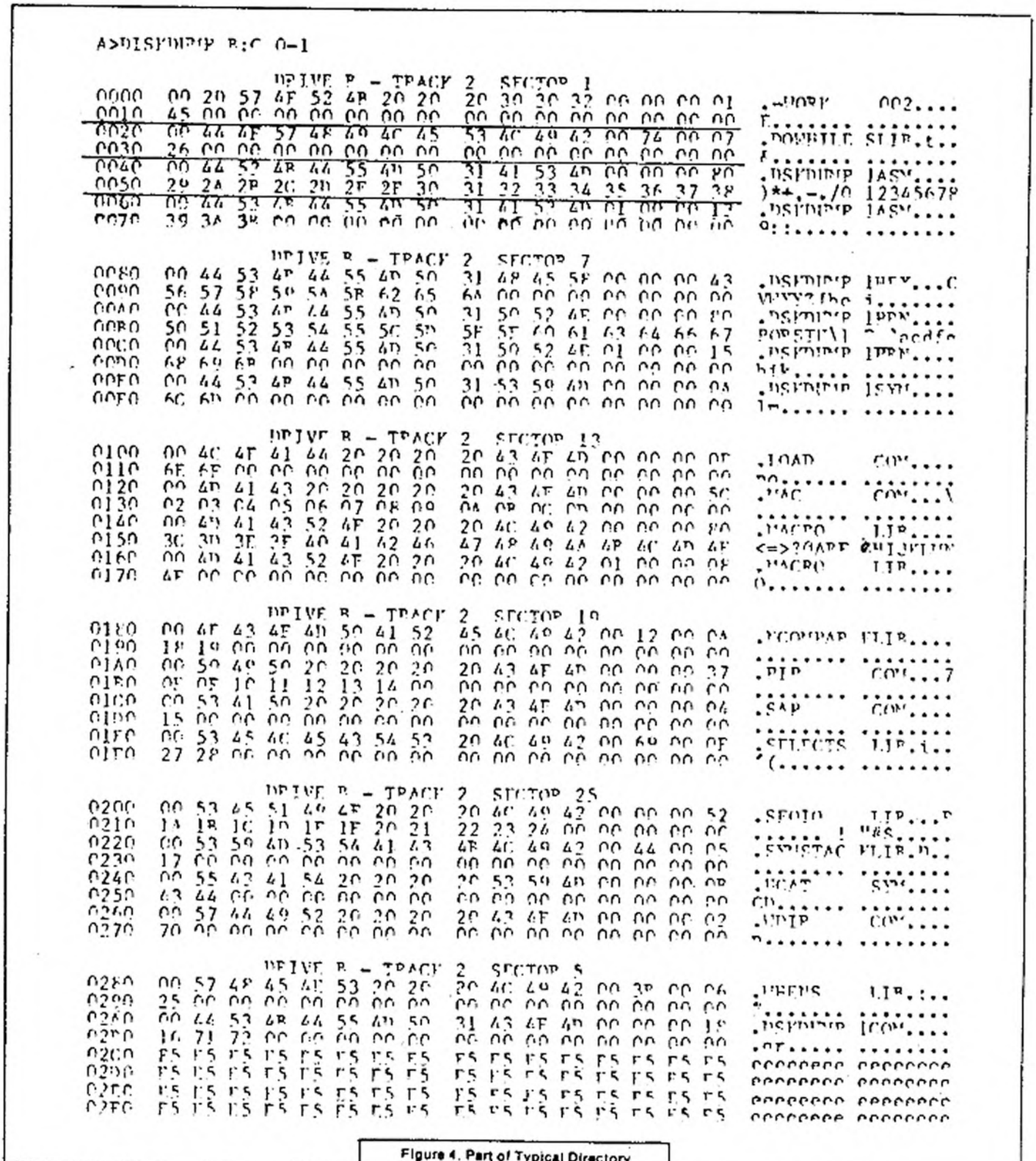
Closing a File

It is not necessary to close a file if ONLY read operations were performed on it. This is because reading alone does not change either the Allocation Map or the directory entries for the file. Closure is highly desirable, however, to maintain upward compatibility of the user program with revisions later than 1.4 of CP/M. In a multi-user system, for example, the file would have to be closed before any other user program could access it.

Further, if any type of writing was done, the space allocations for the file were probably changed, and must be written back to the disk to ensure integrity of the data. The Close function copies the current FCB to the matching directory entry, if one exists, or to the first free slot in the directory area if the current FCB describes a new Extent. Then the allocation bit map is written back to the disk area on which it resides, and the entire updated directory is written out to the first sixteen sectors of Track 2.

To close a file, do the following:

- Load register C with the function code (16=10H=Close).
- Load Register Pair DE with the address of the FCB for the file.
- CALL BDOS (entry point is at 005H).



```
A>DISKDUMP B:0-1

                    DRIVE B - TRACK 2  SECTOR 1
0000  00 20 57 4F 52 4B 20 20  20 30 30 32 00 00 00 01  . WORK      002....
0010  E5 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  e...............
0020  00 44 4F 57 4B 49 4C 45  53 4C 49 42 00 76 00 07  .DOWNFILE SLIB.v..
0030  26 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  &...............
0040  00 44 53 4B 44 55 4D 50  31 41 53 4D 00 00 00 80  .DSKDUMP 1ASM....
0050  29 2A 2B 2C 2D 2E 2F 30  31 32 33 34 35 36 37 38  )*+,-./0 12345678
0060  00 44 53 4B 44 55 4D 50  31 41 53 4D 01 00 00 13  .DSKDUMP 1ASM....
0070  39 3A 3B 00 00 00 00 00  00 00 00 00 00 00 00 00  9:;.............

                    DRIVE B - TRACK 2  SECTOR 7
0080  00 44 53 4B 44 55 4D 50  31 4B 45 58 00 00 00 43  .DSKDUMP 1KEX...C
0090  56 57 58 59 5A 5B 62 65  6A 00 00 00 00 00 00 00  VWXYZ[be j.......
00A0  00 44 53 4B 44 55 4D 50  31 50 52 4E 00 00 00 80  .DSKDUMP 1PRN....
00B0  50 51 52 53 54 55 5C 5D  5E 5F 60 61 63 64 66 67  PQRSTU\] ^_`acdfg
00C0  00 44 53 4B 44 55 4D 50  31 50 52 4E 01 00 00 15  .DSKDUMP 1PRN....
00D0  6B 69 6B 00 00 00 00 00  00 00 00 00 00 00 00 00  kik.............
00E0  00 44 53 4B 44 55 4D 50  31 53 59 4D 00 00 00 0A  .DSKDUMP 1SYM....
00F0  6C 6D 00 00 00 00 00 00  00 00 00 00 00 00 00 00  lm..............

                    DRIVE B - TRACK 2  SECTOR 13
0100  00 4C 4F 41 44 20 20 20  20 43 4F 4D 00 00 00 0F  .LOAD    COM....
0110  6E 6F 00 00 00 00 00 00  00 00 00 00 00 00 00 00  no..............
0120  00 4D 41 43 20 20 20 20  20 43 4F 4D 00 00 00 5C  .MAC     COM...\
0130  02 03 04 05 06 07 08 09  0A 0B 0C 0D 00 00 00 00  ................
0140  00 4D 41 43 52 4F 20 20  20 4C 49 42 00 00 00 80  .MACRO   LIB....
0150  3C 3D 3E 3F 40 41 42 46  47 48 49 4A 4B 4C 4D 4E  <=>?@ABF GHIJKLMN
0160  00 4D 41 43 52 4F 20 20  20 4C 49 42 01 00 00 08  .MACRO   LIB....
0170  4F 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  O...............

                    DRIVE B - TRACK 2  SECTOR 19
0180  00 4F 43 4E 4D 50 41 52  45 4C 49 42 00 12 00 0A  .OCNMPAR LIB.....
0190  1B 1C 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01A0  00 50 49 50 20 20 20 20  20 43 4F 4D 00 00 00 37  .PIP     COM...7
01B0  08 09 0A 0B 0C 0D 0E 00  00 00 00 00 00 00 00 00  ................
01C0  00 53 41 50 20 20 20 20  20 43 4F 4D 00 00 00 04  .SAP     COM.....
01D0  15 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01E0  00 53 45 4C 45 43 54 53  20 4C 49 42 00 60 00 0F  .SELECTS LIB.`..
01F0  27 28 00 00 00 00 00 00  00 00 00 00 00 00 00 00  '(..............

                    DRIVE B - TRACK 2  SECTOR 25
0200  00 53 45 51 49 4F 20 20  20 4C 49 42 00 00 00 52  .SEQIO   LIB...R
0210  1A 1B 1C 1D 1E 1F 20 21  22 23 24 00 00 00 00 00  ......  !"#$.....
0220  00 53 59 53 49 53 54 41  43 4C 49 42 00 44 00 05  .SYSTAC  FLIB.D..
0230  17 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0240  00 55 43 41 54 20 20 20  20 53 59 4D 00 00 00 0B  .UCAT    SYM.....
0250  43 44 00 00 00 00 00 00  00 00 00 00 00 00 00 00  CD..............
0260  00 57 44 49 52 20 20 20  20 43 4F 4D 00 00 00 02  .WDIR    COM.....
0270  70 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  p...............

                    DRIVE B - TRACK 2  SECTOR 31
0280  00 57 48 45 4E 53 20 20  20 4C 49 42 00 3B 00 06  .WHENS   LIB.;..
0290  25 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  %...............
02A0  00 44 53 4B 44 55 4D 50  31 43 4F 4D 00 00 00 1B  .DSKDUMP 1COM....
02B0  16 71 72 00 00 00 00 00  00 00 00 00 00 00 00 00  .qr.............
02C0  E5 E5 E5 E5 E5 E5 E5 E5  E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
02D0  E5 E5 E5 E5 E5 E5 E5 E5  E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
02E0  E5 E5 E5 E5 E5 E5 E5 E5  E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
02F0  E5 E5 E5 E5 E5 E5 E5 E5  E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
```

Figure 4. Part of Typical Directory

Files of type .COM and .HEX do not contain any built-in end-of-file (EOF) marker. If BDOS starts to process a Read request and finds that the count in the NR field of the FCB is greater than the count in the RC field, the request is aborted and a completion code of 1 (read past end of file) is returned. A transient program may use this indication to break out of a data transfer loop; more usually, such a loop is initialized to read only the number of sectors specified by the RC field.

ASCII files of types ASM, TXT, DOC, etc. can also use the above methods. However, the transient programs that process such files (assemblers, editor, text formatters, etc.) expect to find at least one ctrl-Z (1AH) code after the CRLF of the last record in the file. Some programs fill all unused space in the last sector with this code. The EOF marker is not recognized by BDOS, but it acts as a signal to the transient not to read any further sectors, and to ignore the first EOF marker and all subsequent bytes in the buffer.

When random file access is in progress, the user program should always check the completion code returned by each Read or Write request, because BDOS distinguishes between an attempt to read a sector beyond the true end of file (code 1), and an attempt to read a sector which is within the file area but has no data in it (code 2). The latter condition could occur while building a tree, or while using hashed key techniques.

Deleting or Renaming a File

A user program can delete or rename a file by function calls to BDOS. The delete procedure is:

- Place the name and type of the file to be deleted in bytes 1 through 11 of an FCB (bytes are numbered from 0).
- Load Register C with the function code (19=13H=Delete).
- Load Register Pair DE with the address of the FCB.
- CALL BDOS.

No information is returned by BDOS after a Delete request.

The rename procedure is:

- Place the old name and type of the file to be renamed in bytes 1 through 11 of the FCB.
- Place the new name and file type in bytes 16 through 26 of the FCB.
- Load Register C with the function code (23=17H=Rename).
- Load Register Pair DE with the address of the FCB.
- CALL BDOS.

If BDOS finds a directory entry matching the filename and type in FCB bytes 1-11, it changes these to the filename and type specified in FCB bytes 16-26 and returns the byte address (within the sector) of the changed entry, in Register A. If no matching entry is found, BDOS returns 255 (0FFH) in Register A. The user program should check the completion code and take appropriate action if the renaming was not successful. After a successful renaming, a Close request must be issued for the file under the new name, otherwise the modified directory will not be written to the disk.

The CP/M Connection

Chris Terry

Implementing the IOBYTE Function — Part 3

The CP/M System Alteration Manual (page 15) notes that "...the user can optionally implement the IOBYTE function which allows reassignment of physical and logical devices." Unfortunately, the clues to the procedure are scattered through the Facilities Manual, the System Alteration Manual and the Interfacing Manual, and no examples are given.

Why, in practice, would we want to change the active peripherals? We might, for example, have both a dot-matrix printer (on a parallel port) and a daisy-wheel printer (on a serial port); the IOBYTE function allows us to use the dot-matrix printer for numeric output, but to switch to the daisy-wheel for correspondence. Again, if we normally use an electronic keyboard and VDM as the console, but also have a keyboard/printer serial terminal such as a Teletype or Diablo or TI Silent 700, we can switch all console functions to the serial terminal whenever we wish, and switch them back when desirable.

Logical Devices

The ability to perform this switching implies that we have a logical I/O system in which each kind of I/O operation is performed by a separate logical device - that is, a software routine which controls the flow of data, and may do some formatting and CRC generation or checking, but does not directly talk to a physical I/O device.

Communication between the logical device and a physical device takes place through two intermediaries: a logical driver, which is permanently associated with the logical device, and a physical driver that is permanently associated with a particular physical device (see Figure 1). In the distribution version of CP/M, the logical and physical drivers are one and the same; that is, each logical device is permanently linked to one, and only one physical device.

However, when the IOBYTE function is implemented, the logical and physical drivers are separated. The logical driver then consists of a switching mechanism that allows its associated logical device to be linked to any one of four physical drivers (and their associated physical devices). The IOBYTE itself is part of this switching mechanism.

CP/M contains four logical devices. For convenience, they are named:

1) CON: 2) RDR: 3) PUN: 4) LST:
(The colons (:) are part of the names.)

The CON: device provides slow-speed communication between the operator and the operating system. It has three logical drivers: CONST, which checks the character ready/not ready status of the currently assigned console input device; CONIN, which fetches a single character from the console input device, and CONOUT, which outputs one character to the currently assigned console display device.

The RDR: device is for input only, from mass storage devices such as a paper tape reader, a cassette playback, a card reader, a badge reader, etc. It has one logical driver, called READER.

The PUN: logical device is for output only to paper tape, cassette recorder, etc. It complements the RDR: device. It has one logical driver, called PUNCH.

The LST: device is for output only. It is not used by the facilities built into the CCP, though it can be linked in tandem with the console display (ctrl-P toggles this link on and off). It is meant for directing the output of application programs to a printer or to mass storage devices other than the disk subsystem. It has one logical driver, called LIST.

Logical Drivers and the IOBYTE

In the distributed system, which does NOT have the IOBYTE function implemented, the logical drivers

actually contain the physical drivers. This means that each logical device is linked to one, and only one, peripheral.

When the IOBYTE function is implemented, this situation changes. The physical device drivers become separate routines (TTYIN, TTYOUT, etc.). The logical drivers then become selection routines, each of which may select one out of four possible physical drivers according to the code found in the corresponding section of the IOBYTE.

The IOBYTE is located at 0003H, and is divided into four 2-bit sections (see figure 2), each of which is associated with one of the logical devices. The 2-bit code (00, 01, 10, or 11) found in any given section of the IOBYTE selects one of the four physical drivers that can legally be associated with that particular logical device. Figure 2 also shows the names associated with the codes for each logical device. It is important to note that from the viewpoint of the switch mechanism in the logical driver, only the codes themselves matter. The names are merely identifiers of the legal codes in each section of the IOBYTE, and only become useful when the STAT utility is used to change the contents of the IOBYTE - that is, to assign a new peripheral to a logical device.

There are many possible ways of implementing the selection mechanism. A neat and straightforward solution can be found in a program called VBIOS31, by Jeff Kravitz, which is contained in Volume 1 of the CP/M Users' Group library. Each logical driver has the form shown in figure 3, except that after the LDA IOBYT instruction, the LIST driver has two RLC instructions, the PUNCH driver has four RLC instructions and the READER driver has two RRC instructions. The effect of these is to shift the bits of interest into bit positions 0 and 1 of the A register.

IOBYTE AT 0003H

Bit Position-->	7 6	5 4	3 2	1 0
Logical Dev.-->	LST:	PUN:	RDR:	CON:
BINARY DEC	Device names known to PIP & STAT			
00 0	TTY:	TTY:	TTY:	TTY
01 1	CRT:	PTP:	PTR:	CRT:
10 2	LPT:	UP1:	UR1:	BAT:
11 3	UL1:	UP2:	UR2:	UC1:

Figure 2. Device Selection Codes in IOBYTE

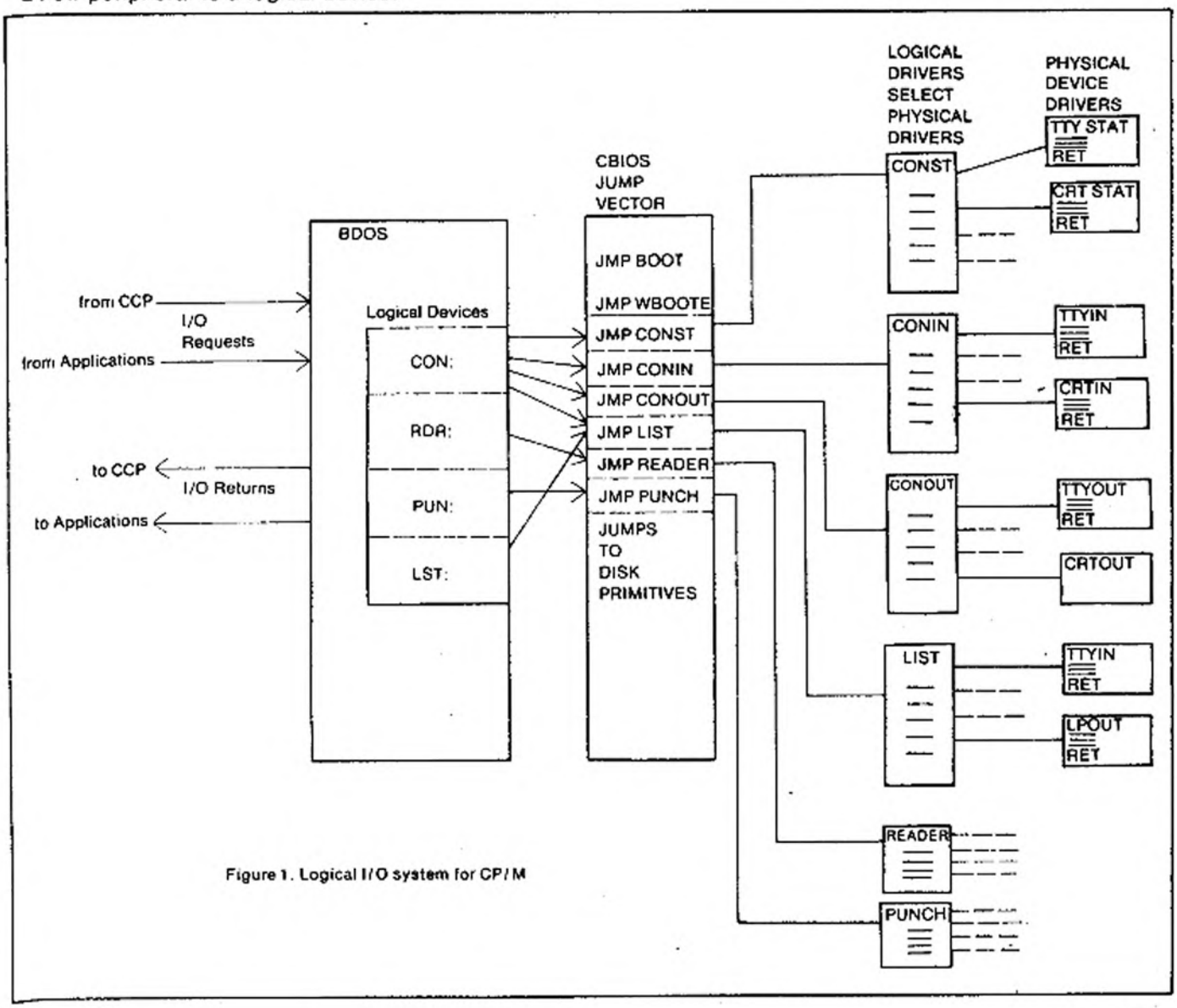


Figure 1. Logical I/O system for CP/M

The CALL to the common I/O Dispatcher (IOCAL) puts the address of the first entry in the table of physical drivers on the stack as the Return address, although it will not be used as such. IOCAL's job is to find which table entry to use, and then to branch to the address contained in the entry. To do this, it uses the IOBYTE code as an offset to be added to the address of the first table entry. The original IOBYTE code ranges from 0 through 3; however, each table entry is two bytes long, and therefore our offset must be doubled so that its possible values are 0, 2, 4, or 6. This is done by the single RLC at the start of IOCAL. Now we set bits 0 and 3 through 7 of the A register to zero with the ANI 6 instruction, which leaves the absolute value of our doubled code in the register to be used as the offset.

The XTHL instruction saves the current contents of the HL register pair on the stack and brings what was on the top of the stack (the address of the first table entry) into HL. To this (after saving the contents of DE) we double-add our offset by clearing D, moving the offset from A into E, and doing a DAD D. The HL register pair now points to the table entry containing the address of the desired physical driver. The next five instructions bring the driver address itself into HL and restore the original contents of DE. The XTHL again swap HL and the top of the stack, so that the physical driver address goes on the stack and the original contents of HL are restored. Finally, the RETurn instruction pops the driver address off the stack into the Program Counter, and we start executing the selected driver. The RETurn instruction at the end of the driver itself passes control back to whichever routine requested the I/O operation.

Thus, every I/O call, whether to BDOS or directly to any one of the logical drivers, causes the IOBYTE to be inspected and control to be passed to the physical driver specified in the appropriate section of the IOBYTE.

Other Considerations

The LST:, PUN:, and RDR: are one-way logical devices, and assigning a new physical device to one of them does not affect either of the other two. The only restriction is the obvious one that it is useless trying to obtain input from an output-only device, and vice versa. Care must be taken, however, in the assignments to CON:, on a two-way logical device. Every assignment to this device changes all three of the associated physical drivers simultaneously; that is, the status driver, the character input driver and the character output driver. The branch tables for these drivers must be set up so that a mistaken reassignment

```
CONIN:  LDA     IOBYTE  ;Gets the complete IOBYTE
        (RLCs or RRCs   ;as needed to shift code into
                ;bits 0-1 of A. None needed for CON:)
        CALL    IOCAL   ;Puts the address of CITBL on stack
CITBL:  DW      TTYIN
        DW      CRTIN
        DW      RDRIN
        DW      UC1IN

IOCAL:  RLC             ;Double the code bits of interest
        ANI     6       ;Mask out all other codes
        XTHL            ;Save HL, get address of XXTBL
        PUSH    D
        MOV     E,A     ;Put doubled code in E
        MVI     D,0     ;and clear D
        DAD     D       ;Add doubled code to XXTBL address
                        ;to find address of required entry
        MOV     A,M     ;Get low byte of entry
        INX     H       ;Now point to high byte of entry
        MOV     H,M     ;and put it into H
        MOV     L,A     ;Put the low byte into L
        POP     D       ;Restore DE
        XTHL            ;Put entry address on stack, restore HL
        RET             ;Pop entry address into PC to start
                        ;executing the required driver (TTYIN)

TTYIN:  ---
        ---
        ---
        RET             ;Returns control to original caller
```

Figure 3. Typical Code for one Logical Driver (CONIN), an associated Physical Driver (TTYIN), and the common I/O Dispatcher (IOCAL).

command does not cause loss of all communication between the operator and the operating system.

Suppose we have a keyboard and VDM as our standard console, a serial CRT terminal as the alternative device and do not intend to use the BAT: (input from RDR:, output to LST:) or UC1: (user-defined) logical devices. Then in our CONST and CONIN logical drivers, the first table entry will branch to TTYST and TTYIN drivers, and in CONOUT the first table entry will branch to the VDM driver software. The second table entry in CONST and CONIN will branch to the CRTST and CRTIN routines, and the second entry in the CONOUT table will branch to CRTOUT. For the third (BAT:) and fourth (UC1:) table entries, we have two possibilities:

a. In each table, make the 3rd and 4th entries the same as the first. This will automatically default them to the standard device.
b. Put branches to an error handling routine. This might merely be a null input routine that returns a NULL (00) and a null output routine that copies C into A and then returns; or the error handling routine might include an error message.

Initialization. As we have seen, communication between the operator and the computer is now totally dependent upon having the correct code in bits 0 and 1 of the IOBYTE. At power-on time, this byte contains a random bit pattern. It is therefore essential that the CP/M Coldstart portion of the boot procedure be modified to include proper initialization of the IOBYTE. If the assignments are set up so that the first entry in each table (code 00) sets up our normal system configuration, the initializing code merely clears the A register (XRA A) and deposits this 00H in IOBYTE (STA IOBYT).

Device Assignment from the Console

The STAT utility has the ability to list the legal device assignments for each of the four logical devices, to list the current assignments, and to change the current assignments. STAT does not know (or care) how the logical driver tables are set up in the CBIOS; it is concerned only with examining the contents of the IOBYTE at location 0003H, reporting what it finds there, and changing specific bits in the IOBYTE while leaving the remainder untouched.

To obtain the list of legal assignments, we type the command:

A>STAT VAL:

which generates the response:

```
CON: = TTY: CRT: BAT: UC1
RDR: = TTY: PTR: UR1: UR2
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

If we wish to know the current device assignments, we type the command:

A>STAT DEV:

If the IOBYTE contains the bit pattern 10 11 01 00, the response will be:

```
CON:IS TTY:
RDR: IS PTR:
PUN: IS UP2:
LST: IS LPT:
```

Now, if we wish to change the reader assignment from PTR: (which could be a fast paper-tape reader) to UR1: (which could be a cassette), we type the command:

A>STAT RDR:=UR1:

STAT would then change the code in bits 2 and 3 of the IOBYTE from 01 to 10. All subsequent requests for Reader input would then access the cassette instead of the paper-tape reader.

If we want to change more than one assignment, we can put up to four such commands on the same line, separating them with commas, e.g.:

A>STAT RDR:=UR2:,LST:=TTY:

STAT will detect and deny any request to assign an input physical driver to an output logical device, or to assign device names that are unknown to it, with the error message:

INVALID ASSIGNMENT

Space on the System Tracks

There is one last item which we must take into account: the space on the system tracks (0 and 1) that is available to expand the CBIOS. The standard CBIOS begins at 3E00 in a 16K system or 7E00 in a 32K system. Thus, we have 512 bytes available for all CBIOS functions (including the disk primitives). If our expanded CBIOS (with the new IOBYTE function) requires more than 512 bytes, then we shall have to move CP/M downward by 1K in order to fit the new CBIOS between the top of the BDOS and the top of available memory. This creates a space of 512+1024=1536 bytes above the top of the BDOS. We cannot use all of this space, however, since only 9 sectors (1152 bytes) are available for the CBIOS on System Track 1. We must therefore ensure that the last byte of object code in our expanded CBIOS has a memory address no greater than XE7F (where CBIOS starts at XA00). There is nothing to prevent us from using the space XE80 thru XFFF as scratchpad memory.

If memory space is tight, and we only require (say) 640 bytes for the new CBIOS, we could move CPM downward by only one page (256 bytes). However, a shift of less than 1K will make computation of the ORG address of CBIOS and of the offset less convenient.

The CP/M Connection

Chris Terry

How to Use the CP/M Facilities in Your Own Applications Programs — Part 4

Previous articles in this series described how the CP/M file management system works, and how the I/O system can be enhanced to give a choice of peripherals. This article discusses how the facilities provided by CP/M can be used in your own application programs, with particular emphasis on portability.

PORTABILITY CONSIDERATIONS

Avoiding System Incompatibility

Let me say at the start that, for programs intended to be run *only* on one's own system, there is no reason not to make use of any and all the facilities of CP/M. Disk I/O primitives and peripheral drivers in the CBIOS may be invoked with subroutine calls—but at a price. The price paid is that the program may not run on any system that uses a different disk format. Prime examples of this are the SAP (Sort And Purge directory), WDIR (Wide DIRectory), and XDIR (extended attribute DIRectory) programs in Volumes 19, 4, and 24 of the CP/M Users' Group library. These work like a charm on systems using CP/M Version 1.4 with 8" IBM-compatible (soft-sectored) drives, and are extremely useful utilities. But they all use direct calls to disk primitives in the CBIOS that assume 77 tracks, 26 sectors per track, and 128 bytes per sector. They do not run on any other disk system, nor under CP/M Version 2. SAP, in particular, destroys the directory if run on a double-density system. All three of these utilities need extensive recoding for double-density or hard sectoring.

There are some advantages in using direct calls, in spite of David E. Cortesi's vigorous letter of protest in the February 1981 issue of *Interface Age*. He rightly points out that direct calls to console drivers lose the facilities provided by CCP (e.g., the ability to suspend/continue an operation with a Control-S). However, there are times when one wishes to use control characters in a manner that conflicts with CCP's use of them; one must then either change the design to use different characters, or employ a direct call to the console driver. Also, direct calls can avoid the need to save registers not used by the primitives.

In general, portability demands that an application program:

1. Perform ALL input and output operations via calls to BDOS, not direct to the CBIOS drivers.
2. Contain all other routines required by the application.
3. Provide adequate stack space of its own, storing the CP/M stack pointer on entry and restoring it on exit.
4. Save all registers containing significant data before each call to BDOS, and restore them on return from the call.

Performing all I/O via BDOS ensures that these operations are system-independent. Regardless of system version or size, a BDOS call takes the function code in the C register and a character or buffer address in the DE register pair; and the BDOS entry point is at 0005H.

Point No. 2 may seem obvious, but sometimes gets forgotten when adapting a private program for publication. I have a 5K ROM monitor that combines and enhances the most useful features of Roger Amidon's Apple monitor and the old Processor Technology Software Package; it contains excellent buffer scanning and code conversion routines that I frequently call from CP/M application programs. But these are peculiar to my system, and when someone would ask me for a copy of one of my utilities, I sometimes would forget to extract one of them. Nowadays I keep the most useful ones in a .LIB file for easy inclusion in programs to run on other systems.

Point No. 3 is important and is often overlooked. CP/M has a pretty good stack, but a program that does a lot of PUSHing and POPping when subroutines are already nested to considerable depth can exhaust the CP/M stack and crash. I have encountered this problem several times, chiefly because my programming style leans toward a separate subroutine for each logical

function; my main program loop is nothing but a series of subroutine calls to major functions, and those in turn call nested subroutines to perform subfunctions. My programs may run a little more slowly because of this—but then they are I/O limited rather than CPU limited, anyway; I certainly find that this style makes the logic easy to follow when I want to modify a program six months after it was written. Further functions can be added or algorighms changed with very little trouble.

Point No. 4 is also important. The BDOS uses all registers, and even a simple call for console input or output can destroy data in BC or HL if these registers are not saved. If a macro assembler is used, code Save and Restore macros and use them before and after every call to BDOS, remembering that single-byte results are returned in the A register, and double-byte results in BC.

Avoiding Size Incompatibility

CP/M systems come in all sizes from 16K (minimum) up to 64K (maximum without memory Bank switching). Regardless of system size, Page 0 (locations 0000 through 00FFH) and 22 pages at the top of memory are reserved for the use of CP/M. Thus, the area available for application programs and their work space varies from 40 pages (10K) in a minimum system to 248 pages (62K) in a maximum system. This variability has implications both for application programs which make only BDOS calls, and for those which make direct calls to CBIOS routines.

Finding Available Memory Size. Programs which perform extensive searches, such as text editors and formatters, and database update and retrieval programs, run faster if the data to be processed (or a substantial portion of it) is available for processing in memory. Also, telecommunication programs communicating with remote computers generally require large buffers to avoid interruption of data transfers on the line by disk accesses. Programs of this type need to know the last memory location available to them, so that they can define large buffers and workspace without encroaching upon CP/M. Locations 5, 6, and 7 in Page 0 always contain a jump instruction to the BDOS entry point; thus, the last available memory address can be found by the sequence:

```
LHLD 0006H
DCX H
SHLD MEMTOP
```

where MEMTOP is a 2-byte storage location within the application program.

Finding The CBIOS Jump Vector. Programs which make direct calls to CBIOS routines need to know the locations of these routines. One way of doing this is to include in the application source code equate statements that give the addresses contained in the CBIOS jump vector. This is an unsatisfactory method, however, since it ties the executable .COM program to one particular CP/M size; if the system is changed the equates must also be changed and the application program reassembled. It is better to define the CBIOS locations dynamically upon entry to the application program; the program will then run correctly on any system that uses a similar disk format.

There are numerous ways of locating the CBIOS routines at run time; all depend on the fact that CP/M places a jump instruction in location 0000, followed by the address of the Warm Boot routine, which is the second item in the CBIOS jump vector. This vector contains the following items:

```
JMP   BOOT      ;Arrive here from cold start load
JMP   WBOOT     ;Arrive here for warm start
JMP   CONST     ;Check for console input character ready
JMP   CONIN     ;Read character from console input
JMP   CONOUT    ;Write character to console display
JMP   LIST      ;Write a character to list device
JMP   PUNCH     ;Write a character to Punch device
JMP   READER    ;Read a character from Reader device
JMP   HOME      ;Move to track 00 on selected drive
JMP   SELDSK    ;Select a disk drive
JMP   SETTRK    ;Set track number on selected drive
JMP   SETSEC    ;Set sector number on selected track
JMP   SETDMA    ;Set address of disk I/O buffer
JMP   READ      ;Read selected sector
JMP   WRITE     ;Write selected sector
```

Once the address of the jump to WBOOT is known from locations 0001-0001, the application program can compute the address of any other CBIOS routine. The simplest way of doing this is to create an identical jump vector within the application program, as follows:

```
;*********************************************************
;GETVEC.LIB,  A routine to obtain
;CBIOS routine addresses for a local jump vector.
;From CP/M Users' Group Library, Volume 1.
;*********************************************************
;
CRT   100H
;
VECTRS:   JMP   GETVEC
          DS    42          ;This space will contain the local
                            ;jump vector after execution of GETVEC.
;---------------------------------------------------------------
WBOOT:    EQUVECTRS+3       ;Do NOT remove any items
CONST:    EQUVECTRS+6       ;from this list, or addresses
CONIN:    EQUVECTRS+9       ;will not match those in the
CONOUT:   EQUVECTRS+12      ;CBIOS jump vector.
LIST:     EQUVECTRS+15
PUNCH:    EQUVECTRS+18
READER:   EQUVECTRS+21
HOME:     EQUVECTRS+24
SELDSK:   EQUVECTRS+27
SETTRK:   EQUVECTRS+30
SETSEC:   EQUVECTRS+33
SETDMA:   EQUVECTRS+36
READ:     EQUVECTRS+39
WRITE:    EQUVECTRS+42
;---------------------------------------------------------------
;
GETVEC:   LXI   D,WBOOT     ;Set Destination to start of local vect
          LHLD  1           ;Get start address of CBIOS vector.
          MVI   B,42        ;Set byte count (14 jumps X 3).
GETVE1:   MOV   A,M         ;Get a byte from CBIOS vector
          STAX  D           ;and copy it to the local vector.
          INX   H           ;Bump the
          INX   D           ;pointers,
          DCR   B           ;check the byte count,
          JNZ   GETVE1      ;and loop till done.
;
;Warm local vector complete, fall through
;into main body of application program.
;*********************************************************
```

BUFFEREDI/O

Buffered Console I/O

CP/M provides facilities for buffered I/O as well as single-character I/O. Calls to BDOS for buffered I/O have the same form as other BDOS calls—that is, the appropriate function code is placed in the C register and the starting address of the buffer in the DE register pair. A subroutine call to BDOS at the standard entry point (0005H) then initiates the operation.

Buffered console input is used by CP/M for commands and their arguments, using the default I/O buffer at 80H. On input, characters are echoed to the console

display device and then accumulated in the buffer, starting at TBUF+2, and the byte count at TBUF+1 is updated. TBUF contains a constant representing the maximum buffer length (80 characters for the CCP).

Accumulation ends when a Carriage Return is entered, and control is returned to the calling program. Application programs can also use this facility; they can use the default buffer at TBUF, or can define a buffer of up to 256 characters (including the maximum length byte, the current byte count, and the Carriage Return terminator).

Buffered console output is used mainly for messages. The print buffer function is placed in the C register and the address of the string to be printed in the DE register pair. There is no limit on the string length; multiple lines can be printed in one operation by including Carriage Return/Line Feed (ODH, OAH) line separators. Printing stops when a Dollar sign ($, 24H) is encountered, and control is returned to the calling program. This function is strictly for ASCII printable characters; there are no facilities for converting hexadecimal or BCD numbers to ASCII.

Buffered Disk I/O

All data transfers to and from successfully opened disk files take place via a 128-byte (single-density) or 256-byte (double-density) buffer. BDOS is told the starting address of this buffer by a SETDMA function call; all subsequent read and write operation use this one-sector buffer until the address is changed by another SETDMA call to BDOS. If an application program does not execute a SETDMA call to BDOS, reading and writing takes place through the default buffer (TBUF) at 80H. The application program is responsible for filling the buffer before issuing a Write, and for extracting the data from the buffer after a Read; routines to do this are discussed later in the commented listings. The read and write calls must supply the address of a File Control Block; if reading/writing is sequential, BDOS updates the NR (Next Record) byte in the FCB after each operation.

In many applications it is desirable to read more than one sector before processing the data. In such cases, the application must define a buffer of appropriate size (4K is common, and editors may create buffers of 20K or more). A SETDMA call to BDOS establishes the address for the first read, and a further SETDMA augments the buffer address by one sector length after each read. The same procedure is used for writing multiple sectors to the disk.

The CP/M Connection

Chris Terry

A Real Application — Part 5

What, Another BLEEDIN' PRINT UTILITY? Grrrr!

That's right! But it's not that big, has some new features that you may like, and gives me a chance to talk about the hooks into CP/M, as well as a few other things.

All those print utilities in the CP/M Users' Group library are fine as far as they go, and work well, but they have one big failing and some minor inconveniences. The big failing is that they follow too strictly Humpty Dumpty's prescription for a good story: "Begin at the beginning, go on till you come to the end, and then stop." That's alright for a first printing, but what about the times when we have drastically changed one routine on page 18 of 24? Can we print just that page? No way! I got so frustrated with this that the first change in my new version is the ability to print from any page to any later page or, of course, from page one to the end.

The print utilities' minor inconveniences are:

- They are set for 60 lines on a page 66 lines deep. Well, I have a Hytype printer that likes an elite wheel, with a page 88 lines deep. So the next change was to vary the number of lines printed. You can set the first line and the last line, so if you want you can print twelve lines in the middle of each page.
- Error messages are not helpful, usually consisting of just one word "ERROR." Yech! Was it a file open error, a read error, or what? Let's at least distinguish between those two.
- The very first operation is a form feed. This makes no sense to me. If we didn't tear off the last printout, we are already at top of form. If we did, we had to move the paper up to bring the inter-page perforations clear of the printer, so we have to wind it up again to the top of the next page— and the program now skips yet another page. My version prompts you to set a new page and initialize the printer, waits till you have done so, and then starts with the title of page one.

I am sure some of you will think these are idiotic gripes. But in these days of cheap memory, there's no excuse for being chintzy about prompts, error messages and operating procedures, when a few more lines of code will make the program "user-friendly" (if you'll forgive the Madison Avenue hype). This is something I feel strongly about. I spend a lot of time at my work cooking up data entry programs for non-technical people who need explicit (but not verbose) directions from the program. If they don't get these, they infallibly find new ways to defeat the data validations, and sometimes even manage to give the Operating System hiccups.

The Structure Of NEWPRINT

Before I get into specifics on the CP/M hooks and on some of the routines, I want to talk about the general structure of NEWPRINT. I am not pushing "structured programming," since nineteen programmers will have seventeen different ideas on just what that is. There is even a myth going around that assembly language is unsuitable for structured programs. It's true that it is much easier to write spaghetti-bowl assembly code than to write easily understandable and maintainable code. But any structure is better than none; I strongly believe that each of us should develop for ourselves a style that is logically sound and visually striking, and that we should follow our guidelines in all our programs. It is not necessary to be rigid, that is why I use the term "guidelines" instead of "rules." My own guidelines are set out below, and may make it easier for you to follow what's going on in NEWPRINT. My particular style has grown out of the fact that I'm a technical writer, and I have come to look on the language and the visual impact of the listing as a precision instrument for communicating the logic.

History and Operation

I always start a source listing with a brief introduction giving the date of coding, and subsequent modifications. I like to have a brief statement of what the program does, and a summary of any special features or hardware and software constraints. If these grow too large, they should go into a separate .DOC text file, but the source listing should have a reference to such a file. And finally, it should have brief operating instructions, or at least how to enter the program and perform its major functions.

Constants and External References

All constants and the addresses of external routines called by the program should be defined by EQU statements and grouped right at the beginning or end of the program. My preference is to put them at the beginning,

because they are usually known by the time coding starts; then if I forget what I called something, I can go back to the Equate section and look it up. It helps me to be consistent as I am working, and so avoid those "UNDEFINED" error messages at assembly time.

It is obvious that the addresses of external routines must be defined here, but what about constants? Yes indeed, ALL constants too! To my mind, dropping numeric constants into a program is like dropping one's contact lenses onto a patch of wet pebbles—you have to find them again before they are any good to you. And when modifying some value, it's all too easy for me to miss one occurrence of it. But when it comes time to change the program all occurrences of a symbolic constant can be changed simultaneously merely by redefining it in a single EQU statement up front. In NEWPRINT, for example, if you think my buffers are too long or too short, you can change the space alloted merely by redefining STAKLEN or NUMSEC. If you don't have elite type, redefining MINL and MAXL to 4 and 63 will give you 60 pica lines per page and take care of all references to these values at assembly time.

One joy of assembly language is that there is a comment field on every single line, just waiting to be filled up.

The Main Program and the Subroutines

I have found that I understand my programs better, six months after I have written them, if I have a setup section (which may call subroutines), a main program which is little more than a loop consisting of subroutine calls, and a subroutine for each important logical function. Where I have to jump out of the main line, I prefer to jump forward to a separate block which performs its function and then returns to the appropriate point. I am well aware that this is not always efficient, either in keystrokes or in execution time, but then most of my programs are I/O bound (I don't do much number crunching)so efficiency is not of prime importance. On the other hand, this way of working allows code lines that perform a particular function to be separated out and made visible by blank or comment lines. If you look at the main printing loop from 0121 to 0138, you will see what I mean. The TBLP tab expander and the PCLF end-of-line code were originally embedded in the main line between the end of LOOP (at 0132) and the beginning of LOOPX (at 0133). They were perfectly functional, but it was not so easy to spot what they were for, especially as they were very sparsely commented.

A logical function should have only one entry and one exit. This is not easy, and sometimes not feasible in assembly language, so I'm not rigid about it, but when it can be done it makes changes much easier. I prefer to follow logic downward, even if this results in jumps to another jump which then takes us back to the top of a loop, rather than have several exits that go backward—and all too often land us right in the spaghetti bowl.

Comments

One joy of assembly language is that there is a comment field on every single line, just waiting to be filled up. None of this nonsense about a GOTO or a GOSUB having to be the last statement on a line (which precludes comments). Let us be generous with comments, but let us also make them work for us. Better no comment than:

```
SUB  B  ;Subtract content of B reg.
```

What, in heaven's name, have we got in the B register? Or in A? It's no more work, and a lot more helpful, to say:

```
SUB  B  ;BOTM-TOPM
```

when we know what BOTM and TOPM contain (from previous comments or definitions).

CP/M Hooks (And Traps)

Console Functions

Now we are getting to the meat and potatoes. At 0115 we have a call to SETUP, which uses the CP/M Write Console Buffer function to output prompts, and the Read Console Buffer function to collect the responses. Write Console is simple to use; just put the function code (09H) in C and the address of the message in DE. Writing continues until a $ sign is found; thus, a multiline message that includes CR-LF sequences can be written with a single call to BDOS; MARMSG (the prompt for line numbers) and PAGMSG (the prompt for page numbers) are examples of this.

Read Console is a little more tricky. You have to define a buffer (in this case, ABUF) and put the maximum length (in bytes) in the first byte. ZBUF (0383) does this, and also initializes the rest of the buffer to 00. To collect keyboard input, we put the function code (0AH) in C, the address of the first byte of ABUF (which has the maximum length value) in DE, and call BDOS. A CR-LF is issued to the console and BDOS waits for keyboard input. As each key is struck the character is echoed to the console and put in ABUF, starting at ABUF+2, until CR is hit; the CR does not go into the buffer but writes CR-LF to the display. The second byte now contains the number of characters just received. This is useful if you don't want to clear the buffer every time you use it.

You can, if you wish, use the standard Console/Disk buffer (TBUF) from 80H through OFFH. However, my own tests (CP/M 1.4) on this showed the message starting at TBUF+4—TBUF contained a 2-byte length and TBUF+2 a 2-byte count, which is not kosher according to the manual. I have not figured this out yet, and I was getting unpredictable results, so I kept things simple and used ABUF for responses. When scanning, I first check ABUF+2 for a slash (indicating default values). If it's not a slash, ADEC scans ASCII decimal digits from ABUF+2 until it finds a non-decimal delimiter, converts the decimal number to binary, and stashes it away. Then ADEC is called a second time to convert the second number. ABUF is ten bytes long to hold two 3-digit numbers and a delimiter. ADEC can convert up to five digits to a 16-bit binary number.

One other Console function is used, Interrogate Status. You will find this in the PBYT routine starting at 01C6. This is the routine that outputs bytes from the file to the list device, provided that the Print Enable flag is set to indicate that the current page is within the printing range you gave it. Interrogate Status does not look at the

DE register pair; just put OBH in the C register and call BDOS. It is used at 01E4 after printing each byte to see if an abort has been signalled by pressing some key. But BEWARE. This function drove me crazy for nearly a week. The original code (composed for CP/M 1.3) had CPI 0 instead of ANI 1 to test the response, but the contents of bits 1 through 7 of the A register are undefined on the return from BDOS; only bit 0 is significant—0 for no input, 1 if any key was pressed. The CPI 0 had worked in previous versions of the program, but now some bit other than 0 was consistently set, so that the program always aborted before printing anything. At first I thought I had an unbalanced stack, but examination of the stack area showed that the last return address was 01E9, and consultation with a friend reminded me to read the manual again; it clearly specifies bit 0 as the test criterion. *Always* read the manual carefully, and *never* take anybody else's coding as gospel truth.

Always read the manual carefully, and never take anybody else's coding as gospel truth.

An error in opening the file causes a branch to NOFILE at 01F7. This piece of code (a CALL instruction followed by non-executable message text) has caused raised eyebrows and baffled mutterings in some quarters. In fact, it is a perfectly standard method of passing the address of a parameter list to the destination routine. Don't think of it as a subroutine CALL expecting a RET, think of it as a simulated JS (Jump and Store) instruction that pushes an address on the stack and expects the destination routine to recover that address with a POP instruction. If you look at ERR (0165) you will see that the address of the "CANNOT OPEN FILE" message is popped off the stack into DE, the Write Console Buffer code is put in C, and BDOS is called to write the message. On return from BDOS, a jump to EREXIT picks up the CP/M stack pointer and reboots the system. Parameter passing methods are discussed in more detail in an article by S. Mazor and C. Pitchford, entitled "Develop Cooperative Microprocessor Subroutines" (Electronic Design, No. 12, June 7, 1978).

Let's take a look at GETBT (0310.) When we first enter the program, the pointer stash INPTR (03C3) is initialized to contain the address of the first byte past the end of the disk buffer. The first call to GETBT sets DE to that address, loads the contents of INPTR into HL, and compares them. Since they are equal, the buffer is obviously empty and control is transferred to FILBUF (032D) which will refill the buffer from the disk. On completion of the Read, INPTR is loaded with the address of the start of the buffer and is incremented each time a byte is taken off the buffer and printed. Thus, each time the buffer is emptied by GETBT, FILBUF replenishes it until End-of-File is reached.

Since we are not using the standard 1-sector buffer at TBUF (80H), we have to tell BDOS where to load memory from the disk. On the first pass through FILBUF, we put the buffer start address in LOADA and in DE, put the SETDMA (or here DEFDMA) code in C and call BDOS. To do the read, we put the Read code in C, the address of the File Control Block in DE, and call BDOS. There are several ways of setting up for multiple sector reads. I have chosen to initialize a counter (CURSEC) in memory to NUMSEC and count it down to zero. After each read, we get the starting point of the completed read from LOADAD, add SECLEN to it (128 for single density, 256 for double density), stash the updated address back in LOADAD and in DE, and call BDOS to update the starting address of the new read.

A good read within the file returns 00 in the A register, and we continue reading until the buffer is full. There are two other possible conditions, handled by CKEOF (0366) if we do not get a 0 (good read) back from BDOS. One of these is the End-of-File condition (1 in the A register); we stop reading, but go back to GETBT to allow printing to continue. When the main loop fetches the first EOF marker (1AH) from the disk buffer, we take a normal exit to CP/M via DONE (0159). The other condition is non-recoverable disk error which returns something other than 0 or 1 in the A register. In that case we drop through CKEOF into RDERR (036B), which is similar to NOFILE. ERR prints the "READ ERROR" message on the console, and EREXIT cleans up and reboots CP/M.

Disk Functions

Three disk I/O functions are used in NEWPRINT: Open File, Set DMA, and Read next sector. The initialization section calls the FOPEN subroutine (at 02E8) to open the file passed as an argument in the PRINT command. To open a file for sequential reading, put 0FH in the C register, the address of the File Control Block (in this case, the standard TFCB at 5CH) in DE, and call BDOS. On return, the A register contains either 0FFH if the file was not found, or the byte address of the FCB containing the directory entry. BDOS transfers the entry from the disk to the specified FCB. A successful open initializes the NR (next record) byte of the FCB, and subsequent Read operations update it.

Writing the List Device

The last CP/M function to be described is Write Character to List Device. This is used in PBYT (01C6) to send characters (fetched out of the disk buffer by GETBT) to the list device. To ouput a character, put the function code (1AH) in C, clear D, put the character itself in E, and call BDOS. The Write List function does not trap or filter characters; it is transparent to the user. All character trapping must be done by the calling program—refer again to LOOP-LOOPX and the associated special processing routines TBLP (Tab expander), PCLF (end-of-line processor), and DONE (normal exit on end-of-file).

With this guide to the structure of the program, I believe that the comments in the listing are adequate to fill in the details. I have tried to be explicit about multi-sector reading from a sequential file. Writing to a sequential file is similar except that the application fills OBUF until OPTR points beyond the end of the buffer; then we do a Write (instead of the Read used in this program).

```
                ;*********************************;
                ;     FILE PRINT UTILITY          ;
                ;*********************************;

                ;       ORIGINAL CODED BY JEFF KRAVITZ AND MODIFIED
                ;       10/15/77 BY A. GOLD FOR <FF> HARDWARE.
                ;       LARGELY RECODED BY CHRIS TERRY, 1/15/81 FOR
                ;       DAISY WHEEL OR SPINWRITER PRINTERS.
                ;This print utility is for use with any CP/M system;
                ;it is an enhancement of the PRNT utility in Volume 1
                ;of the CP/M Users' Group library.  The revision:
                ;       1)  Assumes a printer that responds to Form Feed (0CH);
                ;       2)  Assumes a printer that can be manually set for
                ;           pica or elite type, and that any hardware
                ;           automatic form feed at page end can be turned off;
                ;       3)  Requests line numbers for 1st and last lines
                ;           on each page (on /, defaults to 5 and 84 for elite,
                ;           giving 80 lines with equal 1/2" top and bottom
                ;           margins for elite type);
                ;       4)  Requests numbers of 1st and last pages to be
                ;           printed (on /, prints all to max. 255);
                ;       5)  Requests that paper be set to top of new page and
                ;           any hardware auto form-feed turned OFF.
                ;       6)  Multisector disk buffer to reduce number of disk head
                ;           loads, especially when not printing.

                ;Invoke the utility with the PRINT
                ;command and the name and type of the file to be
                ;printed.  When asked for line numbers or page
                ;boundaries for printing, type the first and last
                ;numbers separated by a space (Page limit is 255)
                ;or a slash to keep defaults.
                ;       A>PRINT file.typ

                ;*********************************;
                ;     MISCELLANEOUS EQUATES       ;
                ;*********************************;

                ;       *** CP/M ADDRESSES***
0000 =          BOOT    EQU     0000H   ;REBOOT ENTRY POINT
0005 =          BDOS    EQU     0005H   ;BDOS ENTRY POINT
005C =          TFCB    EQU     005CH   ;TRANSIENT PROGRAM FCB
                ;
                ;       ***CP/M FUNCTION CODES***
000F =          OPEN    EQU     15      ;OPEN FILE
0014 =          READ    EQU     20      ;READ SECTOR
000A =          RDCON   EQU     10      ;FILL CONSOLE BUFFER FROM KEYBOARD
000B =          CSTAT   EQU     11      ;GET CONSOLE STATUS
0001 =          RDCHAR  EQU      1      ;GET A CHARACTER FROM KEYBOARD
0002 =          WRCHAR  EQU      2      ;WRITE A CHARACTER TO CONSOLE DISPLAY
0009 =          WRMSG   EQU      9      ;WRITE A STRING TO CONSOLE DISPLAY
0005 =          WRLST   EQU      5      ;WRITE A CHARACTER TO LIST DEVICE

001A =          DEFDMA  EQU     26      ;SET DMA (I/O BUFFER ADDRESS)

                ;       ***ASCII CHARACTERS AND OTHER CONSTANTS***
0009 =          TAB     EQU     09H     ;ASCII TAB CODE
000A =          ASLF    EQU     10      ;ASCII LINE FEED
000D =          ASCR    EQU     13      ;ASCII CAR.RET
000C =          FORM    EQU     12      ;ASCII FORM-FEED
001A =          EOF     EQU     1AH     ;END-OF-FILE MARKER
0000 =          PAD     EQU     00      ;SOME PRINTERS PREFER RUBOUTS (FF)

000C =          TOFNULLS EQU    12      ;NULLS AFTER A FORM FEED
0004 =          CRNULLS EQU      4      ;NULLS AFTER A CR

000A =          ABUFLEN EQU     10      ;ENOUGH TO HOLD 2 3-DIGIT NUMBERS & SPACE
0080 =          SECLEN  EQU     128     ;SECTOR LENGTH (IN BYTES)
0008 =          NUMSEC  EQU     8       ;NUMBER OF SECTORS IN THE DISK BUFFER
0400 =          BUFLEN  EQU     SECLEN*NUMSEC   ;LENGTH IN BYTES OF DISK BUFFER
0040 =          STAKLEN EQU     64      ;LENGTH OF STACK AREA;
0005 =          MINL    EQU     5       ;DEFAULT 1ST PRINT LINE (FOR ELITE)
0054 =          MAXL    EQU     84      ;DEFAULT LAST PRINT LINE (FOR ELITE
0001 =          MINP    EQU     1       ;LOWER PRINT RANGE BOUNDARY (DEFAULT)
00FF =          MAXP    EQU     255     ;UPPER PRINT RANGE BOUNDARY (DEFAULT)

0100            ORG     100H

                ;*********************************;
                ;       MAIN LOOP                 ;
                ;*********************************;

                ;
                ;       *** INITIALIZATION ***
                ;
0100 210000     START:  LXI     H,0
0103 39                 DAD     SP      ;PUT THE CP/M STACK POINTER IN HL
0104 22BA05             SHLD    OSTAK   ;AND SAVE IT;
0107 31B605             LXI     SP,STACK;THEN SET UP A LOCAL STACK
010A AF                 XRA     A
010B 32C103             STA     PFLAG   ;CLEAR THE PRINT FLAG.
010E 32BE03             STA     PAGE    ;INITIALIZE PAGE COUNT TO 0.
0111 3C                 INR     A
0112 32BD03             STA     LINE    ;INITIALIZE LINE COUNT TO 1.
0115 CD6B02             CALL    SETUP   ;GET LINES PER PAGE & PRINT PAGE RANGE
0118 CD4902             CALL    PCHEK   ;SEE IF PAGE 1 IS TO BE PRINTED (INCREMENTS COUNT)
                ;
011B CDEB02             CALL    FOPEN   ;OPEN FILE
                ;
011E CD7E01             CALL    TOF3    ;PRINT HEADING ON PAGE 1 (BUT NO FORM-FEED).

                ;********** MAIN PROGRAM & EXITS TO CP/M **************************

                ;               ***Main Printing Loop***
0121 CD1003     LOOP:   CALL    GETBT   ;GET A BYTE
0124 FE1A               CPI     EOF     ;EOF?
0126 CA5901             JZ      DONE    ;YES, GO CLEAN UP AND DO NORMAL EXIT.

0129 FE0A               CPI     ASLF    ;LF?
012B CA3901             JZ      PCLF    ;YES, END OF LINE.
012E FE09               CPI     TAB     ;TAB?
0130 CA4901             JZ      TBLP    ;YES, GO EXPAND TO 8 SPACES
                ;
0133 CDC601     LOOPX:  CALL    PBYT    ;ALPHANUMERIC, SO PRINT BYTE.
0136 C32101             JMP     LOOP

                ;       ***PROCESS LINE FEED***
0139 CD2602     PCLF:   CALL    LINCHK  ;ARE WE AT END OF PAGE (CARRY SET)?
013C F5                 PUSH    PSW     ;SAVE THE FLAGS
013D DC4902             CC      PCHEK   ;IF EOP, ARE WE IN PRINT RANGE?
0140 F1                 POP     PSW     ;GET THE FLAGS BACK;
0141 DC6E01             CC      TOF     ;IF EOP, SET UP TITLE & PAGE NO.,
0144 3E0A               MVI     A,ASLF  ;EXECUTE THE LF AND
0146 C33301             JMP     LOOPX   ;CONTINUE

0149 3E20       TBLP    MVI     A,' '   ;EXPAND TAB CHARACTER
014B CDC601             CALL    PBYT
014E 3ABB03             LDA     COL
0151 E607               ANI     07H
0153 C24901             JNZ     TBLP
0156 C32101             JMP     LOOP

                ;       ***DONE & EREXIT clean up before exit to CP/M******

0159 3E0C       DONE:   MVI     A,0CH   ;FORM FEED ON NORMAL EXIT
015B CDC601             CALL    PBYT
015E 2ABA05     EREXIT: LHLD    OSTAK   ;RESTORE THE
0161 F9                 SPHL            ;CP/M STACK POINTER.
0162 C30000             JMP     BOOT    ;EXIT TO CP/M

                ;       ***Get here on any error call with message address on stack***
0165 D1         ERR:    POP     D       ;GET MSG ADDRESS PUT ON STACK BY CALL
0166 0E09               MVI     C,WRMSG ;WRITE ERROR MSG TO CONSOLE
0168 CD0500             CALL    BDOS
016B C35E01             JMP     EREXIT  ;GO RESTORE CP/M STACK BEFORE ERROR EXIT.
```

```
0001 =          MINP    EQU     1       ;LOWER PRINT RANGE BOUNDARY (DEFAULT)
```

```
016B C35E01             JMP     EREXIT  ;GO RESTORE CP/M STACK BEFORE ERROR EXIT.
```

```
                ;**************END OF MAIN PROGRAM*****************
                ;               SUBROUTINES FOLLOW
                ;
                ;       ***PROCESS END OF PAGE AND NEW TITLE***
016E 3E0C       TOF:    MVI     A,FORM  ;FORM FEED
0170 CDC601             CALL    PBYT
0173 060C               MVI     B,TOFNULLS
0175 3E00       TOF2:   MVI     A,PAD
0177 CDC601             CALL    PBYT

017A 05                 DCR     B
017B C27501             JNZ     TOF2
                ;
017E 21CA03     TOF3:   LXI     H,FMSG  ;POINT TO 'FILE' MESSAGE
0181 CDF101             CALL    PSTRNG  ;PRINT STRING
0184 215D00             LXI     H,TFCB+1 ;POINT TO NAME

0187 0608               MVI     B,8     ;SIZE OF NAME
0189 CDFC01             CALL    PCNT    ;PRINT 8 NAME CHARACTERS
018C 3E2E               MVI     A,'.'   ;PRINT A PERIOD
018E CDC601             CALL    PBYT
0191 216500             LXI     H,TFCB+9 ;POINT TO TYPE
0194 0603               MVI     B,03    ;SIZE OF TYPE
0196 CDFC01             CALL    PCNT    ;PRINT 3 TYPE CHARACTERS
0199 21D303             LXI     H,PMSG  ;POINT TO 'PAGE' MESSAGE
019C CDF101             CALL    PSTRNG  ;PRINT STRING
019F 3ABE03             LDA     PAGE    ;GET PAGE NUMBER
01A2 CD0602             CALL    DEC     ;CONVERT TO DECIMAL
01A5 21C703             LXI     H,DECWRK ;POINT TO DEC STRING
01A8 0603               MVI     B,3
01AA CDFC01             CALL    PCNT    ;PRINT PAGE NUMBER
01AD 3E0D               MVI     A,ASCR  ;PRINT CR
01AF CDC601             CALL    PBYT
01B2 3E0A               MVI     A,ASLF
01B4 CDC601             CALL    PBYT    ;PRINT LF
01B7 3E0A               MVI     A,ASLF
01B9 CDC601             CALL    PBYT    ;AND ANOTHER
01BC E5                 PUSH    H
01BD 21BD03             LXI     H,LINE
01C0 7E                 MOV     A,M
01C1 C603               ADI     3       ;BUMP LINE COUNT FOR HEADER & SPACES
01C3 77                 MOV     M,A
01C4 E1                 POP     H
01C5 C9                 RET


01C6 E5C5       PBYT:   PUSH H ! PUSH B
01C8 47                 MOV     B,A     ;SAVE THE CHARACTER
01C9 3AC103             LDA     PFLAG   ;CHECK TO SEE IF THIS PAGE
01CC B7                 ORA     A       ;IS TO BE PRINTED (FLAG=1);
01CD CAE401             JZ      PBY2    ;IF NOT, SKIP THE PRINTING,
                ;
01D0 C5                 PUSH    B       ;ELSE SAVE THE CHARACTER
01D1 0E05               MVI     C,WRLST ;PUT FUNCTION CODE IN C,
01D3 1600               MVI     D,0     ;CLEAR D,
01D5 58                 MOV     E,B     ;PUT CHAR IN E FOR BDOS,
01D6 CD0500             CALL    BDOS    ;AND DO THE PRINT.
01D9 C1                 POP     B       ;RESTORE THE CHARACTER.
                ;
01DA 78                 MOV     A,B     ;GET CHARACTER INTO A
01DB FE20               CPI     20H     ;& SEE IF IT'S PRINTABLE
01DD DAE401             JC      PBY2    ;IF NOT, DON'T BUMP COLUMN COUNT,
01E0 21BB03             LXI     H,COL   ;ELSE POINT TO COLUMN COUNT
01E3 34                 INR     M       ;AND INCREMENT IT.
                ;
01E4 0E0B       PBY2:   MVI     C,CSTAT ;GET CONSOLE STATUS
01E6 CD0500             CALL    BDOS    ;TO CHECK FOR AN ABORT COMMAND --
01E9 E601               ANI     1       ;LSB SET?
01EB C25E01             JNZ     EREXIT  ;YES, EXIT BUT WITH NO FORM FEED
01EE C1E1               POP B ! POP H
01F0 C9                 RET
```

```
                ;
                ;       ***PSTRNG is needed since CP/M does not have a
                ;       Print List Buffer function corresponding to the
                ;       Print Console Buffer function***
01F1 7E         PSTRNG: MOV     A,M     ;GET BYTE
01F2 FE24               CPI     '$'     ;STRING END?
01F4 C8                 RZ              ;YES, DONE
01F5 CDC601             CALL    PBYT    ;PRINT BYTE
01F8 23                 INX     H       ;BUMP POINTER
01F9 C3F101             JMP     PSTRNG  ;LOOP

                ;       ***PRINT N BYTES FROM BUFFER ADDRESSED BY HL***
01FC 7E         PCNT:   MOV     A,M     ;GET BYTE
01FD CDC601             CALL    PBYT    ;PRINT IT

0200 23                 INX     H       ;BUMP POINTER
0201 05                 DCR     B       ;DECREMENT COUNT
0202 C2FC01             JNZ     PCNT
0205 C9                 RET

                ;       ***CONVERT BINARY PAGE NUMBER TO ASCI DECIMAL***
0206 21C703     DEC:    LXI     H,DECWRK
0209 0E64               MVI     C,100
020B CD1902             CALL    DIGIT
020E 0E0A               MVI     C,10
0210 CD1902             CALL    DIGIT
0213 0E01               MVI     C,1
0215 CD1902             CALL    DIGIT
0218 C9                 RET

0219 3630       DIGIT:  MVI     M,'0'
021B 91         DI0:    SUB     C
021C FA2302             JM      DI1
021F 34                 INR     M
0220 C31B02             JMP     DI0
0223 81         DI1:    ADD     C
0224 23                 INX     H
0225 C9                 RET
```

```
                ;*********************************;
                ;       LINCHK                    ;
                ; Updates line count and checks   ;
                ; for page end. Carry set to      ;
                ; trigger a new page heading.     ;
                ;*********************************;

0226 E5C5       LINCHK: PUSH H ! PUSH B
0228 0604               MVI     B,CRNULLS

                ;       ***AFTER A CR, DELAY PRINTING***
022A AF         LNULLS: XRA     A
022B CDC601             CALL    PBYT
022E 05                 DCR     B
022F C22A02             JNZ     LNULLS
                ;
0232 21BD03             LXI     H,LINE
0235 34                 INR     M       ;BUMP LINE COUNT IN MEM,
0236 46                 MOV     B,M     ;AND GET UPDATED COUNT INTO B.
0237 2B                 DCX     H       ;POINT TO LNCNT
0238 7E                 MOV     A,M
0239 90                 SUB     B       ;SUBTRACT UPDATED COUNT
023A D24102             JNC     LINDON  ;NO CARRY, STILL ON-PAGE.
023D 23         PAGUP:  INX     H       ;CARRY=OFF PAGE, SO
023E 3600               MVI     M,0     ;RESET CURRENT LINE TO 0
0240 37                 STC             ;AND MAKE SURE CARRY IS SET.
0241 21BB03     LINDON: LXI     H,COL   ;RESET COLUMN COUNT TO 0
0244 3600               MVI     M,0
0246 C1E1               POP B! POP H
0248 C9                 RET             ;AND EXIT.
```

```
                ;*********************************;
                ;        PCHEK                    ;
                ; If LINCHK returns with Carry set;
                ; for new page, this routine      ;
                ; updates the page count and      ;
                ; checks to see if page is to be  ;
                ;        printed or not           ;
                ;*********************************;

0249 E5C5       PCHEK:  PUSH H ! PUSH B
024B 21BE03             LXI     H,PAGE  ;POINT TO CURRENT PAGE NUMBER
024E 34                 INR     M       ;AND UPDATE IT.
024F 23                 INX     H       ;POINT TO PAG1 (LOWER BOUND)
0250 46                 MOV     B,M     ;GET START PAGE NO.
0251 3ABE03             LDA     PAGE    ;GET CURRENT NO.
0254 B8                 CMP     B       ;CURRENT-LOWER
0255 D25C02             JNC     UPPER   ;LOWER BOUND OK, GO CHECK UPPER BOUND.
0258 AF         CLEAR:  XRA     A       ;NOT IN PRINT RANGE, SO CLEAR PRINT FLAG
0259 C36502             JMP     FSTOR   ;AND GO STORE IT.
                ;
025C 47         UPPER:  MOV     B,A     ;A STILL HAS CURRENT PAGE, SO SAVE IT.
025D 23                 INX     H       ;POINT TO PAG2 (UPPER BOUND);
025E 7E                 MOV     A,M     ;GET UPPER BOUND INTO A;
025F B8                 CMP     B       ;UPPER-CURRENT
0260 DA5E01             JC      EREXIT  ;IF CARRY, CURRENT IS ABOVE UPPER BOUND
                ;                       ;SO EXIT TO CP/M AT ONCE.  ELSE
0263 3E01               MVI     A,1     ;SET FLAG TO ENABLE PRINTING,
0265 32C103     FSTOR:  STA     PFLAG   ;AND STORE IT.
0268 C1E1               POP B ! POP H
026A C9                 RET


                ;*********************************;
                ;        SETUP                    ;
                ; Prompts for top and bottom line  ;
                ; numbers, gets them from ABUF and ;
                ; stores ;then in TOPM and BOTM.   ;
                ; Then gets and stores page numbers;
                ; for printing range.              ;
                ;*********************************;
026B CD8303     SETUP:  CALL    ZBUF    ;CLEAR THE ABUF (ASCII BUFFER
                ;                       TO HOLD ANSWERS)
026E 0E09               MVI     C,WRMSG
0270 11F603             LXI     D,MARMSG ;PROMPT FOR 80-LINE DEFAULT
0273 CD0500             CALL    BDOS
                ;
0276 0EDA               MVI     C,RDCON ;NOW READ THE ANSWERS
0278 11B103             LXI     D,ABUF  ;INTO THE ABUF
027B CD0500             CALL    BDOS
                ;
027E 3AB303             LDA     ABUF+2  ;LOOK AT THE FIRST CHARACTER
0281 FE2F               CPI     '/'     ;IF ITS A SLASH
0283 CA9B02             JZ      SETUP3  ;GO SET DEFAULT LINES/PAGE,
                ;
0286 01B303             LXI     B,ABUF+2 ;ELSE POINT TO FIRST NUMBER
0289 CD9503             CALL    ADEC    ;CONVERT IT TO BINARY IN HL
028C 7D                 MOV     A,L
028D 32AF03             STA     TOPM    ;AND STORE IT.
                ;                       ;BC NOW POINT TO SEPARATOR SPACE.
0290 03                 INX     B       ;POINT TO 2ND NUMBER,
0291 CD9503             CALL    ADEC    ;CONVERT TO DECIMAL IN HL,
0294 7D                 MOV     A,L
0295 32B003             STA     BOTM    ;AND STORE IT.
0298 C3A502             JMP     LINSET  ;NOW GO COMPUTE PAGE DEPTH
                ;
029B 3E05       SETUP3: MVI     A,MINL  ;SET DEFAULT TOP AND
029D 32AF03             STA     TOPM
02A0 3E54               MVI     A,MAXL  ;BOTTOM LINES ON THE PAGE
02A2 32B003             STA     BOTM

                ;       ***Now compute no. of lines per page***

02A5 21AF03     LINSET: LXI     H,TOPM
02A8 46                 MOV     B,M
02A9 23                 INX     H       ;POINT TO BOTM
02AA 7E                 MOV     A,M
02AB 90                 SUB     B       ;BOTM-TOPM
02AC 3C                 INR     A       ;LNCNT=BOTM-TOPM+1
02AD 32BC03             STA     LNCNT   ;AND STORE IT FOR FUTURE USE

                ;       ***NOW FALL INTO PAGE RANGE ROUTINE***
                ;
02B0 CD8303     SETUP4: CALL    ZBUF    ;CLEAR THE ABUF (ASCII BUFFER)
                ;
02B3 0E09               MVI     C,WRMSG
02B5 116704             LXI     D,PAGMSG
02B8 CD0500             CALL    BDOS    ;PROMPT FOR PAGE PRINTING RANGE
                ;
02BB 0E0A               MVI     C,RDCON ;NOW GET PAGE RANGE
02BD 11B103             LXI     D,ABUF  ;INTO THE ABUF
02C0 CD0500             CALL    BDOS
                ;
02C3 3AB303             LDA     ABUF+2  ;LOOK AT THE FIRST CHARACTER,
02C6 FE2F               CPI     '/'     ;IF ITS A SLASH
02C8 CADD02             JZ      SETUP6  ;GO SET MAX PAGE RANGE,
                ;
02CB 01B303     SETUP5: LXI     B,ABUF+2 ;ELSE POINT TO LOWER PAGE NUMBER
02CE CD9503             CALL    ADEC    ;CONVERT IT TO BINARY IN HL
02D1 7D                 MOV     A,L
02D2 32BF03             STA     PAG1    ;AND STORE IT. B POINTS TO THE SPACE.
02D5 03                 INX     B       ;POINT TO UPPER PAGE NUMBER,
02D6 CD9503             CALL    ADEC    ;CONVERT IT TO BINARY IN HL,
02D9 7D                 MOV     A,L
02DA C3E402             JMP     SETUP7  ;AND STORE IT.

                SETUP6:                 ;SET MAXIMUM PAGE RANGE FOR PRINTING
02DD 3E01               MVI     A,MINP  ;FIRST PAGE
02DF 32BF03             STA     PAG1    ;TO
02E2 3EFF               MVI     A,MAXP  ;LAST POSSIBLE PAGE
02E4 32C003     SETUP7: STA     PAG2
02E7 C9                 RET


                ;*********************************;
                ;          F O P E N              ;
                ;  ROUTINE TO OPEN A DISK FILE    ;
                ;                                 ;
                ;  INPUT:  Set DE to address of   ;
                ;    the FCB. On open error, exit;
                ;    with message.                ;
                ;*********************************;

02E8 0E0F       FOPEN:  MVI     C,OPEN  ;OPEN CODE
02EA 115C00             LXI     D,TFCB
02ED CD0500             CALL    BDOS    ;ISSUE OPEN
02F0 FEFF               CPI     0FFH    ;ERROR?
02F2 CAF702             JZ      NOFILE  ;YES
02F5 AF                 XRA     A       ;CLEAR CARRY
02F6 C9                 RET
                ;
02F7 CD6501     NOFILE: CALL    ERR     ;THE CALL PUTS ADDRESS OF MSG ON STACK.
02FA 0D0A43414E         DB      0DH,0AH,'CANNOT OPEN FILE ',0DH,0AH,'$'


                ;*********************************;
                ;          G E T B T              ;
                ; Routine to read a byte from     ;
                ; IBUF; replenish from disk when  ;
                ; the buffer is empty.            ;
                ; OUTPUT:A =BYTE from disk buffer; ;
                ;*********************************;

0310 21BC09     GETBT:  LXI     H,IBUF+BUFLEN
0313 EB                 XCHG            ;BUFFER END ADDR. IN DE
0314 2AC303             LHLD    INPTR   ;CURRENT POINTER IN HL
0317 CD7D03             CALL    CPHL    ;TEST FOR END OF BUFFER
```

```
02A2 32B403             STA     BUFM
                ;       ***Now compute no. of lines per page***
031A CA2D03             JZ      FILBUF  ;YES, READ
031D 7E         GETB1:  MOV     A,M     ;GET BYTE
031E 23                 INX     H       ;BUMP POINTER
031F 22C303             SHLD    INPTR   ;SAVE POINTER
0322 B7                 ORA     A       ;RESET CARRY
0323 C9                 RET
                ;
0324 21BC05     GETB2:  LXI     H,IBUF  ;RESET BUFFER POINTER
0327 22C303             SHLD    INPTR
032A C31D03             JMP     GETB1   ;CONTINUE
                ;
032D 3E08       FILBUF: MVI     A,NUMSEC ;SET NUMBER OF SECTORS TO READ
032F 32C203             STA     CURSEC  ;AND PUT IT IN CURRENT SECTOR COUNT
0332 21BC05             LXI     H,IBUF
0335 22C503             SHLD    LOADAD  ;INITIALIZE DISK READ ADDRESS
0338 EB                 XCHG            ;PUT THIS IN DE
0339 C34703             JMP     FILBU3  ;AND GO SET INITIAL LOAD ADDRESS.
                ;
033C 2AC503     FILBU2: LHLD    LOADAD  ;GET LOAD POINT OF LAST READ,
033F 118000             LXI     D,SECLEN ;ADD THE SECTOR LENGTH
0342 19                 DAD     D       ;TO THE PREVIOUS LOAD POINT,
0343 22C503             SHLD    LOADAD  ;STASH IT AWAY FOR NEXT TIME AROUND,
0346 EB                 XCHG            ;AND PUT IN DE
                ;
0347 0E1A       FILBU3: MVI     C,DEFDMA ;FOR BDOS
0349 CD0500             CALL    BDOS    ;TO SET THE NEW LOAD ADDRESS.
034C 0E14               MVI     C,READ  ;FUNCTION CODE
034E 115C00             LXI     D,TFCB  ;PASS THE TFCB ADDRESS TO BDOS
0351 CD0500             CALL    BDOS    ;READ SECTOR,
0354 FE00               CPI     0       ;CHECK FOR GOOD READ
0356 C26603             JNZ     CKEOF   ;IF NOT, SEE IF EOF,
0359 3AC203             LDA     CURSEC  ;ELSE GET SECTOR READ COUNT,
035C 3D                 DCR     A       ;DECREMENT IT,
035D CA2403             JZ      GETB2   ;IF 0, WE'RE DONE READING,
0360 32C203             STA     CURSEC  ;ELSE STASH UPDATED SECTOR COUNT,
0363 C33C03             JMP     FILBU2  ;AND GO READ SOME MORE.
                ;
0366 FE01       CKEOF:  CPI     1       ;IF 1 RETURNED, WE'RE PAST EOF
0368 CA2403             JZ      GETB2   ;SO GO PRINT WHAT WE GOT.
036B CD6501     RDERR:  CALL    ERR
036E 0D0A524541         DB      0DH,0AH,'READ ERROR',0DH,0AH,'$'

                ;*********************************;
                ;   MISCELLANEOUS SUBROUTINES     ;
                ;*********************************;

                ;*********************************;
                ;             C P H L             ;
                ;  ROUTINE TO COMPARE HL VS DE    ;
                ;*********************************;

037D 7C         CPHL:   MOV     A,H
037E BA                 CMP     D
037F C0                 RNZ
0380 7D                 MOV     A,L
0381 BB                 CMP     E
0382 C9                 RET
                ;
                ;*****************************************************
                ;ZBUF fills the top bytes of ABUF with 00
                ;and puts the length in the 1st byte for BDOS.
                ;The ZBU1 portion can be used for other buffers if the
                ;top address and fill count are put in DE and B by caller.
                ;
0383 3E0A       ZBUF    MVI     A,ABUFLEN
0385 32B103             STA     ABUF
0388 AF                 XRA     A
0389 11B203             LXI     D,ABUF+ABUFLEN
038C 0609               MVI     B,ABUFLEN-1 ;So as not to clear length byte.
038E 1B         ZBU1    DCX     D
038F 12                 STAX    D
0390 05                 DCR     B
0391 C28E03             JNZ     ZBU1
0394 C9                 RET
                ;
                ;*****************************************************
                ; ADEC fetches up to 5 ASCII decimal digits from the ABUF
                ;addressed by BC and converts them to a 16-bit binary value
                ;returned in HL. Scan stops on a space or 00.
                ;
0395 210000     ADEC:   LXI     H,0
0398 0A         ADE1:   LDAX    B
0399 FE30               CPI     30H
039B D8                 RC              ;DONE IF CHAR < '0'
039C FE40               CPI     40H     ;OR > '9'
039E D0                 RNC
039F 54                 MOV     D,H
03A0 5D                 MOV     E,L
03A1 29                 DAD     H
03A2 29                 DAD     H
03A3 19                 DAD     D
03A4 29                 DAD     H
03A5 D630               SUI     30H
03A7 5F                 MOV     E,A
03A8 1600               MVI     D,0
03AA 19                 DAD     D
03AB 03                 INX     B
03AC C39803             JMP     ADE1

                ;*********************************;
                ;     W O R K  S P A C E           ;
                ;*********************************;
03AF 05         TOPM:   DB      MINL    ;1ST PRINT LINE
03B0 54         BOTM:   DB      MAXL    ;LAST PRINT LINE
03B1            ABUF:   DS      ABUFLEN ;ASCI BUFFER FOR RESPONSES
03BB 00         COL:    DB      0       ;CURRENT COLUMN COUNT
03BC 50         LNCNT:  DB      80      ;LINES PER PAGE
03BD 00         LINE:   DB      0       ;CURRENT LINE COUNT
03BE 00         PAGE:   DB      0       ;CURRENT PAGE NUMBER
03BF 01         PAG1:   DB      MINP    ;LOWER PRINT BOUND
03C0 FF         PAG2:   DB      MAXP    ;UPPER PRINT BOUND
03C1 01         PFLAG:  DB      1       ;PRINT FLAG, 1=PRINT, 0=NO PRINT
03C2 08         CURSEC: DB      NUMSEC  ;CURRENT READ SECTOR COUNT
03C3 BC09       INPTR:  DW      IBUF+BUFLEN ;INPUT POINTER INITIALIZED TO BUFFER END
03C5 BC05       LOADAD: DW      IBUF    ;DISK LOAD INITIALIZED TO START OF BUFFER
03C7 303030     DECWRK: DB      '000'   ;WORKSPACE FOR DECIMAL CONVERSION OF PAGE
                ;
                ;       ***MAIN MESSAGE AREA***
03CA 0D0A46494CFMSG:    DB      0DH,0AH,'FILE: $'
03D3 2020202020PMSG:    DB      '                PAGE $'
03EE 4552524F52ERMSG:   DB      'ERROR',0DH,0AH,'$'
03F6 0D0A446566MARNSG:  DB      0DH,0AH,'Default top & bottom lines are 5, 84, for elite
0428 0D0A               DB      0DH,0AH
042A 456E746572         DB      'Enter top line SPACE bottom line, or a / to keep default
0464 0D0A24             DB      0DH,0AH,'$'
0467 0D0A456E74PAGMSG:  DB      0DH,0AH,'Enter 1st page SPACE last page, or a / to print
049D 0D0A202020         DB      0DH,0AH,'          ...WAIT...'
04B3 0D0A41646A         DB      0DH,0AH,'Adjust paper to 1st print line on new page,'
04E0 0D0A536574         DB      0DH,0AH,'Set printer switches for pica or elite,'
0509 0D0A747572         DB      0DH, 0AH,'turn OFF hardware line counting'
052A 0D0A627574         DB      0DH,0AH,'but initialize form-feed point.',0DH,0AH
054D 5468656E20         DB      'Then hit Return key to start printing.',0DH,0AH,'$'
                ;
                ;       ***STACK AREA***
0576            AREA:   DS      STAKLEN ;Local stack area
05B6 =          STACK:  EQU     $       ;Top of local stack
05B6 0000               DW      0
05B8 0000               DW      0
05BA 0000       OSTAK:  DW      0       ;CP/M Stack pointer storage
05BC            IBUF:   DS      BUFLEN
                ;
09BC                    END     PRINT
```

Running North Star DOS and CP/M Together

Randy Reitz

I have always been interested in CP/M and its dynamic file management system. Last year I started to experiment with the Lifeboat implementation of CP/M for the North Star controller. Using CP/M is a great change from the North Star DOS. The North Star disk operating system (DOS) only performs directory maintenance and low level disk access whereas CP/M has features that should be found in a DOS such as file open, file close, etc. I became interested in how I could use my North Star programs under CP/M as well as use the CP/M editor to prepare text for my North Star Basic programs. The North Star system has some useful programs for poking around and I thought they would be helpful for exploring CP/M. For example, the North Star Monitor program easily dumps and modifies memory, while with the North Star RD command, the contents of a disk can be examined. So the natural outcome of this was to experiment with CP/M using North Star DOS, North Star Monitor and eventually North Star Basic.

Since the CP/M programs FDOS (Basic Input Output System—BIOS plus the Basic DOS—BDOS) and the Console Command Processor (CCP) are in high memory and the North Star DOS and its programs are at 2000H (in the Transient Program Area—TPA), my first idea was to load both systems and switch between them whenever desired. However, I quickly realized that simply running an unmodified North Star DOS in CP/M's TPA did not work well. When I did something in North Star DOS that required a disk access and then returned to CP/M, I would get unpredictable results. Since computers are supposed to be very predictable, I set out to find what was causing the incompatibility between North Star DOS and CP/M.

I didn't have to look too long to find 4 bytes in the North Star DOS that were causing the problem. I could say the problem was really with the North Star Micro-Disk System (MDS) controller. The North Star controller is simplicity itself. You may have noticed there is no "big" LSI chip on the North Star controller board. Everything on the board is simple, ordinary TTL stuff. This simplicity is deceiving since simple hardware usually requires complicated software. Now, I don't want to say that the North Star DOS software is all that complicated; but since the North Star controller board is simple, the North Star software must do more than software that uses a TARBELL controller. For example, there is no way to query the North Star controller board to find out what drive or track is currently selected. This extremely relevant information must be maintained by the software.

Since the controller board is simple, the North Star DOS software must do more than software that uses a TARBELL controller.

Now, about those 4 bytes in the North Star DOS. In locations 2000H through 2002H, the current track number for each drive in the 3-drive North Star system is stored. In location 2003H, the number of the currently selected drive is stored. CP/M on North Star has to keep this same information in software, but since 2000H-2003H is smack in the middle of the TPA, Lifeboat's BIOS keeps this information elsewhere. This is the problem with running two systems together—these bytes need to be synchronized. You can imagine what happens when CP/M's BIOS looks and sees the drive motors are running (which means a drive has been selected) and checks its memory and finds the requested drive (or track) is selected, then proceeds when North Star DOS just finished with a different drive (or track). This condition guarantees unpredictable results.

Fortunately, the solution for this problem is straight forward. The folks at Lifeboat merely lifted the North Star DOS disk drivers that are in the ROM on the controller board and dropped the software unmodified into their BIOS. It didn't take too much work with a disassembler to find where the drivers were in

Randy Reitz, 26 Maple St., Chatham Township, NJ 07928.

Chapter III
CP/M on North Star Systems

Lifeboat's BIOS. Lifeboat tried to discourage me since they inserted an extra byte after each RET and JMP instruction. This drives a disassembler wild; but once I figured out what was going on I could correct for it. It's hard to keep secrets from a good disassembler and a persistant software hack.

The following are the steps required to modify North Star DOS to use the CP/M disk drivers so one set of memory locations are used to keep track of the disk system status. I have been using this "patched" North Star DOS for a while and I can say that it is well behaved.

Lifeboat tried to discourage me since they inserted an extra byte after each RET and JMP instruction. This drives a disassembler wild; but once I figured out what was going on I could correct for it. It's hard to keep secrets from a good disassembler and a persistent software hack.

The first 4 steps set up the environment to patch the North Star DOS. The Dynamic Debugging Tool (DDT) program in CP/M is well suited for this work. The Assembly and List (disassemble) commands are useful and the ASCII interpretation in the Dump command is also helpful.

1. Cold start (boot) CP/M for North Star (Lifeboat CP/M 1.4).

2. Cold start North Star DOS release 4.0 or 5.1S.

3. Insert the CP/M disk in drive A (North Star drive 1) and give the North Star DOS command JP 0. (i.e. get back to CP/M).

4. Issue the CP/M DDT command (i.e. start the CP/M dynamic debugging tool).
   Now comes the point of this whole exercise.

5. Patch the North Star DOS in RAM at the following addresses:

4.0	5.1S	Change to	Comment
2205H		MVI M,0E5H	for N* IN command
2206H		LXI H,0850DH	for IN command
2438H	2381H	CALL BIOS+480H	use CP/M drivers
2497H	2410H	LDA BIOS+5FAH	use CP/M current
24A2H	2428H	STA BIOS+5FAH	drive selected

The value of BIOS above is calculated as MSIZE* 1024-512 where MSIZE is the size of your CP/M in kilobytes. For example, I have 56K of memory so the largest CP/M I can run is 52K since the Lifeboat BIOS is 4K larger than the regular CP/M BIOS. Hence BIOS = 52*1024-512 = 52736 (CE00H). These changes can be made easily with the DDT A(assemble) command.

6. Patch the North Star I/O area to use FDOS.I/O functions. I will discuss below a suggested patch to use.

7. M2000, 2A00,100 (move North Star DOS to 100H).

8. Enter the following DOS mover program with the A(assemble) command:

```
100H    JMP     8E0H    use N*DOS buffer area
8E0H    LXI     H,BIOS+566H   for mover program
8E3H    LXI     D,906H  patch BIOS for 7 bytes
8E6H    MVI     C,7     with code at 906H
8E8H    LDAX    D
8E9H    MOV     M,A
8EAH    INX     H
8EBH    INX     D
8ECH    DCR     C
8EDH    JNZ     8E6H
8F0H    LXI     H,100H  now move N*DOS to
8F3H    LXI     D,2A00H proper location
8F6H    LXI     B,E400H this is -2A00H
8F9H    MOV     A,M
8FAH    STAX    D
8FBH    INX     H
8FCH    INX     D
8FDH    INX     B
8FEH    MOV     A,B
8FFH    ORA     A
900H    JNZ     8F9H
903H    JMP     2A8AH   start N*DOS V5.1S
906H    MVI     A,FE    look for -1 command
908H    CMP     C       this is 5.1S single
909H    JZ      2740H   density disk init
90CH    NOP
```

9. Exit DDT with a control-C and execute the CP/M command "SAVE 10 NSTAR.COM".

The reason for moving the North Star DOS program to 100H is to create the NSTAR command on the CP/M disk. When in CP/M, typing NSTAR will load the 20 records (10 pages) saved above and then the CCP jumps to location 100H. At 100H is a JMP 8E0H that executes the patch and mover program (step 8). This program patches 7 bytes in the CP/M BIOS to accomodate the new North Star DOS 5.1S command (-1) for single density disk initialization. This command was added to the DCOM entry in 5.1S so a disk could be initialized without using a buffer outside the North Star DOS.This BIOS patch isn't needed if you are using release 4.0, but it can still be put in since release 4.0 will not recognize the -1 command. After patching BIOS, the program moves the North Star DOS to 2000H and starts the DOS at the point in the cold start routine that calls TINIT. This will execute whatever initialization routine you have provided as well as check the "auto" start byte. Hence, you could have the DOS do a "GO BASIC, 2" immediately. I located this patch and mover program in the middle of the disk buffer in the North Star DOS. The jump at 903H to start the North Star DOS should be JMP 208AH if you are using release 4.

When running North Star DOS with the CP/M disk driver, you should not have any problems if you are careful not to disturb the CP/M FDOS (BDOS + BIOS) and the bytes at 0-3 that contain the JMP WARM to get back to CP/M and the IOBYTE.

One motivation for running North Star DOS and CP/M together is to use the North Star monitor and

North Star Basic programs to experiment with CP/M. Another reason is to use North Star Basic to move files from a North Star disk to CP/M and vice versa. Following is a segment from a North Star Basic program that allows North Star Basic access to the CP/M FDOS facilities. The most important feature of this program is the assembly routine that provides the North Star Basic interface to CP/M. The segment of the program that does this is given below:

```
 2  DEF FNC(N,D)
 3       FILL 64,N
 4  RETURN CALL (65,D)
 5  FNEND
 6  DATA 58,64,0,79,205,5,0,95,111,201,0
 7  F=92
 8  FILL F,0
 9  FOR I=1 TO 11
10       FILL F+I,ASC("?")
11       READ X \ FILL 64+I,X
12  NEXT
13  R=FNC(13,F)
```

North Star Basic provides a method for accessing user written assembly language subroutines by using the CALL command. The CP/M FDOS can be considered such a subroutine. So a North Star Basic program can use FDOS to do the disk functions necessary to manipulate CP/M files. The FDOS cannot be called directly by North Star Basic since the conventions for passing arguments in the 8080 registers don't agree for North Star and CP/M. Hence, another small assembly language program is needed to adjust the 8080 registers.

The North Star Basic CALL command can contain one or two arguments. The first argument is a numeric value between 0 and 65535 that is the decimal value of the memory address at the beginning of the assembly language subroutine. If a second argument is used, it will be converted to an integer value between 0 and 65535 and placed in the DE register pair. Since the CALL command is a Basic function, it will return a value. The value returned is an integer from 0 to 65535 that represents the value in the HL register pair when the assembly language subroutine returns.

The CP/M FDOS entry point is at address 5. CP/M requires a function number in the C register. Any address information that the CP/M function requires should be in the DE register pair. CP/M returns single byte results in the A register. If a double byte result is returned, the high order byte is in the B register and the low order byte is in the A register. Now that the register conventions are known, it is simple to write a assembly language program that North Star Basic can use to access CP/M FDOS:

```
ADDRESS-CODE  LABEL    OPCODE  ARGUMENTS

40H 00        FUNCTION DB              0
41H 3A4000    DOCPM    LDA             FUNCTION
44H 4F                 MOV             C,A
45H CD0500             CALL            5
48H 60                 MOV             H,B
49H 69                 MOV             L,A
4AH C9                 RET
```

The first byte of this program is used to pass the FDOS function value. This value is put in the C register and FDOS is called. When CP/M returns, the return code is put in the HL register pair. Now look at the multi-line function in the North Star Basic program (lines 2-5). This function expects two arguments, N and D. The first argument, N, is the CP/M FDOS function number and is "poked" into the "FUNCTION" byte in the assembly language program above. The next argument, D, is used for address information. The CALL to the interface program is made with D as the second argument. Recall that the North Star Basic CALL command will put the second argument in the DE register pair, just where CP/M FDOS expects the argument to be,so no adjustment is required. The Basic "FNC" function expects the interface program to be at address 64. Address 64 stores the CP/M function value for the interface program and the CALL is made to address 65. CP/M provides a 16-byte space starting at 64 for the user's CBIOS. If your CBIOS doesn't use these 16-bytes, you can use it for the interface program. Finally, the interface program sets up the HL register pair with the CP/M return code and returns to North Star Basic.

The other important part of the Basic program is line 6, that contains the assembly code for the interface program, and lines 8-12 that put the assembly code into memory starting at address 65. This program is using the CP/M default FCB that is at address 5CH (92 decimal).

Now I'll show you a North Star Basic program that will move a text file from North Star disk to a CP/M disk in drive A (North Star drive 1).

This program begins the same way as the last one. On line 13 the name of the North Star file is requested. If the file does not exist, the name is requested again. Next, the CP/M filename is requested. This name must be less than 12 characters to be valid. Lines 18 to 22 move the CP/M filename from the string C$ to the FCB. Notice that a "." is removed and the CP/M file type is loaded into "FT" field of the FCB. Lines 23-25 set up the CP/M file. If desired, this section of the program could detect if the CP/M file already exists, and if so, request permission to delete it. The CP/M file will be created on drive A. Line 27 opens the North Star file.

The file is transferred one byte at a time in the main loop (lines 28 to 39). Each CP/M sector of 128 bytes is loaded into the default buffer (lines 28 to 34) and then written to the CP/M disk (line 37). If more data remains (test in line 39) the main loop continues. Finally, the CP/M file is closed in line 40.

This program expects the North Star file to contain text that is separated by carriage returns. The program inserts a line feed character after each carriage return so the CP/M editor can be used. The North Star end-of-file is an SOH character (ASCII 1). When this is found, the CP/M end-of-file SUB character (ASCII 26) is substituted. The record loop from line 28 to 34 could be changed to accomodate any North Star data file format you desire. The program ends in line 43 by returning to CP/M. I included this to show that since the CP/M warm start entry doesn't require any data in the 8080 registers, the interface program is not required.

```
 1  REM MOVE N* TEXT FILE TO CP/M
 2  DEF FNC(N,D)
 3       FILL 64,N
```

```
 4  RETURN CALL (65,0)
 5  FNEND
 6  DATA 58,64,0,79,205,5,C,96,111,201,0
 7  F=92 \ D=0 \ B1=0 \ W=0 \ DIM C$(16)
 8  FILL F,0
 9  FOR I=1 TO 11
10      FILL F+I,32
11      READ X \ FILL 64+I,X
12  NEXT
13  INPUT "N* FILENAME= ",I$
14  T=FILE(I$) \ IF T>0 THEN 16
15  PRINT I$," -- NOT FOUND" \ GOTO 13
16  INPUT "CP/M FILENAME= ",C$
17  IF LEN(C$)>12 THEN 46
18  FOR I=1 TO LEN(C$)
19  IF C$(I,1)="." THEN 21
20  FILL F+D+I,ASC(C$(I,1)) \ GOTO 22
21  D=8-I
22  NEXT
23  R=FNC(13,0) \ REM RESET CP/M
24  R=FNC(19,F) \ REM DELETE CP/M FILE
25  R=FNC(22,F) \ REM CREATE CP/M FILE
26  T$="CREATE" \ IF R>=128 THEN 44
27  OPEN 40%T,1$,L \ L=2*L
28  FOR I=128 TO 255
29  IF B1=13 THEN 33 \ REM END-OF-LINE
30  READ #0,&B1 \ PRINT CHR$(B1),
31  FILL I,B1 \ IF B1>1 THEN 34
32  FILL I,26 \ EXIT 35 \ REM END-OF-FILE
33  FILL I,10 \ B1=0 \ PRINT \ REM ADD LINE FEED
34  NEXT
35  PRINT
36  PRINT "WRITTING CP/M RECORD # ",EXAM(F+32)
37  R=FNC(21,F)
38  T$="WRITE" \ IF R>0 THEN 44
39  W=W+1 \ L=L-1 \ IF L>0 AND B1<>1 THEN 28
40  R=FNC(16,F)
41  T$="CLOSE" \ IF R=255 THEN 44
42  PRINT "TRANSFER COMPLETE,",W," CP/M RECORDS"
43  PRINT "RETURNING TO CP/M" \ W=CALL(0)
44  PRINT "CP/M ERROR ON ",T$,
45  PRINT "RETURN CODE =",R \ STOP
46  PRINT "CP/M FILENAME TOO LONG"
47  STOP
48  END
```

This is a small example to show what can be done with North Star Basic by using the CP/M FDOS. It isn't difficult to modify this program to transfer files in either direction. So you might suspect that you could use the CP/M editor to prepare the text for a North Star Basic program and then transfer the program to North Star DOS. However, getting the text of the program into North Star Basic is a little tricky.

As a final topic, let me discuss the possibilities that arise when the CP/M FDOS facility is used to implement the North Star input/output routines. The North Star DOS provides a one-page (256 bytes) block at the end of the DOS to carry out four I/O functions:

COUT	—character output
CIN	—character input
TINIT	—terminal initialization
CONTC	—control-C detection.

At entry to the COUT and CIN routines, the A-reg contains a number that represents the device the routine should use to do the I/O. The CP/M FDOS facility uses a code in the C-reg to indicate the function requested. So, one consideration is simply to set up the information in the proper 8080 registers. For example, here is what the North Star COUT routine could look like:

```
                ORG     2900H
        ;
        ;       N* I/O USING CP/M BIOS
        ;
COUT:   PUSH    B       ;B-reg contains char
        ORA     A       ;to output, A-reg is
        MVI     C,2     ;0 for console so use
        JZ      $+5     ;FDOS function 2
        MVI     C,5     ;function 5 otherwise
        LDA     ECHO    ;check if this character
        CMP     B       ;was just read with CIN
        CNZ     DOCPM   ;output character if not
        MVI     A,0FFH  ;reset ECHO flag
        STA     ECHO
        POP     B
        MOV     A,B     ;N* expects char in A-reg
        RET
```

The B-reg contains the character to output when the COUT routine is called. Since CP/M FDOS expects to find the character in the E-reg when function code 2 (output to console) or code 5 (output to list) is used, the DOCPM routine will make the adjustment. The only other consideration is that FDOS automatically "echoes" characters typed so the COUT routine should not. Therefore, the COUT routine compares the character it is about to output with the last character received by CIN.

Next, consider the character input routine:

```
CIN:    PUSH    B       ;can't destroy any regs
        MOV     C,A     ;save input device number
        LDA     NOFILE  ;check if an "input file"
        ORA     A       ;is active
        JZ      READCPM ;take all input from file
                        ;if active
READEV: MOV     A,C     ;restore input device
        ORA     A       ;input from console if 0
        MVI     C,1     ;use FDOS function code 1
        JZ      $+5     ;for console, code 3
        MVI     C,3     ;otherwise
        CALL    DOCPM
        STA     ECHO    ;set ECHO flag
        POP     B
        RET
```

Again, this routine is straightforward. The feature added to normal character input is the capability to read a CP/M file. In the implementation of North Star DOS for CP/M that I have been presenting, the North Star DOS exists as a CP/M command (COM) file. When the CP/M console command program (CCP) receives a string that does not start with the name of any built-in command, the CCP assumes that there is a file on the currently logged-in disk with the name given and an extension of COM. If this is so, the CCP loads the contents of the file into the TPA and sets up the default buffer at 80H with the remaining characters that were typed before the carriage return. CP/M programs usually understand these "arguments" to be CP/M file name(s). Hence, the North Star DOS can be considered a CP/M command that executes in the TPA and will accept a CP/M file name as an argument.

When the North Star DOS begins execution, the terminal initialization routine first gets control. Since CP/M has been running, the terminal doesn't need to be initialized. This routine can be used to check if a CP/M file name has been passed as an argument. For example:

```
TINIT:  MVI     A,0FFH  ;initialize some flags
        STA     CASE    ;my upper/lower case flag
        STA     ECHO
        STA     NOFILE  ;assume no CP/M file
        OUT     0FFH    ;IMSAI front panel lights
        LDA     BUFF    ;look in default buffer
        ORA     A       ;for a CP/M file name
        RZ              ;all done if no file
        LXI     D,FCB   ;prepare to "open" file
        MVI     C,15
        CALL    FDOS
        CPI     255     ;check if successful
        RZ              ;return if no good
        XRA     A       ;else indicate a CP/M
        STA     NOFILE  ;file is active
        STA     FCBCR   ;start at record 0
        MVI     A,80H   ;force file read routine
        STA     IBP     ;to get a new sector
        RET
```

If the North Star DOS was "called" by the CCP with a file name as an argument, the TINIT routine will open the file and if the open is successful, TINIT will reset the NOFILE flag so CIN is forced to read characters from the given CP/M file.

Notice the CIN routine above will jump to the READCPM routine if a CP/M file is active. This routine follows:

```
READCPM:PUSH  B         ;save all registers
        PUSH  D
        PUSH  H
        MVI   C,11      ;check if any key has
        CALL  FDOS      ;been hit on keyboard
        RRC             ;abort CP/M file if so
        JC    FINIS
SKIPLF:LDA    IBP       ;get address of next
        CPI   80H       ;char in input buffer
        CZ    DISKR     ;read another sector if
        MOV   E,A       ;required - set up (DE)
        MVI   D,0       ;to offset of char in
        INR   A         ;buffer - update input
        STA   IBP       ;buffer pointer
        LXI   H,BUFF    ;address of default buf
        DAD   D         ;(HL) now points to next
        MOV   A,M       ;character
        CPI   10        ;look for line feed
        JZ    SKIPLF    ;skip line feeds
        CPI   26        ;look for CP/M eof
        JZ    FINIS     ;terminate CP/M file
        POP   H
        POP   D
        POP   B         ;BC pushed in CIN
        POP   B         ;restore registers and
        RET             ;return char to NS
```

When a CP/M file is active, all input requested by North Star will be taken one character at a time from the CP/M file until the end of file is reached. Subsequent input will then be taken from the input device specified in the A-reg when CIN is called. The READCPM routine uses two subroutines:

```
DISKR: LXI    D,FCB     ;read the next sector
       MVI    C,20      ;of the CP/M file into
       CALL   FDOS      ;the default buffer
       CPI    0         ;check for read error
       RZ               ;return if no problem
       POP    H         ;clear return address
FINIS: MVI    A,0FFH    ;set NOFILE flag
       STA    NOFILE
       LXI    D,FCB     ;close CP/M file
       MVI    C,16
       CALL   FDOS
       POP    H         ;restore registers
       POP    D
       POP    B
       JMP    HEADEV    ;return to CIN routine
DOCPM: PUSH   D         ;do FDOS function
       PUSH   H
       MOV    E,B       ;adjust register
       CALL   FDOS
       POP    H
       POP    D
       RET
```

The DOCPM routine is used by COUT and CIN to execute the selected FDOS function. The only tricky code above is that when DISKR returns successfully, the A-reg is 0 so IBP will be properly initialized for the new sector.

The only other routine required in the North Star DOS I/O is the control-C detection routine. Here it is:

```
CONTC: MVI    C,11      ;see if a key has been
       CALL   FDOS      ;hit on the keyboard
       ANI    1         ;return with Z-flag
       XRI    1         ;reset if no ^C
       RNZ
       MVI    C,1       ;a key has been hit
       CALL   FDOS      ;get character
       CPI    3         ;look for ^C
       STC              ;tell CP/M char was read
       RET
```

These routines use the following symbols:

```
ECHO    DB      0       ;don't ECHO character
NOFILE  DB      0       ;no CP/M file when 0
IBP     DB      0       ;input buffer pointer
CASE    EQU     005F9   ;subject for future
FDOS    EQU     5       ;CP/M entry point
FCB     EQU     5CH     ;default FCB address
BUFF    EQU     80H     ;default buffer addr
```

At this point you might question the usefulness of this discussion. I mentioned above that I could use the CP/M text editor to prepare North Star Basic programs. With this driver for the North Star DOS I/O, you can do the following:

1. Prepare the text of a North Star Basic program using the CP/M editor.
2. Make the first line of this file the North Star command "GO BASIC, 2".
3. Type the CP/M command "NSTAR < filename .TXT".
4. Sit back and watch North Star Basic read in the program you have prepared.

The North Star DOS and any programs running under it will treat the CP/M file as a command file. The file will be read until an end-of-file condition is encountered; then all input will be taken from the device specified in the A-reg when CIN is entered.

This completes my discussions of using North Star DOS and CP/M together. If you would like to try this, but don't want to do it yourself, for $15 I will supply a diskette containing the NSTAR command and some North Star Basic programs to demonstrate what can be done. You must have the Lifeboat CP/M 1.4 for single density North Star.

Patching a CP/M Diskette on a North Star System

Tom Wiens

Every computer system manual at some point includes a pointed warning—always back up you diskettes! But after logging months of work without a fatal error, I became lax and not a little bit careless, until the inevitable happened: late one night I finished off an additional ten pages of writing on a word processor running under CP/M, and began to write to disk—ERROR: DISK FULL was the program's response. All my attempts to salvage the situation only got me in more trouble. Finally, in desperation, I rebooted without ending the run or closing the file in the hope that some temporary files could still be found on the diskette and used to reconstruct the text. But the CP/M DIR command produced no evidence that any part of my creation was alive or well in any form.

Having found in the past that erased CP/M files could easily be salvaged (see *The CP/M Connection*, Part II, in Vol. 1/ No. 5), and groaning at the prospect of many hours wasted recreating lost ideas, I turned immediately to North Star DOS and the North Star monitor to have a look at the CP/M directory. On my version (Lifeboat Associates' single-density CP/M 1.4 for North Star diskettes), the latter is stored as 8 blocks beginning at disk address 30, which can be read into memory using the DOS RD command, and then examined and modified with the aid of the monitor. In this instance I found that the directory contained one "live" old version of the document I was working on (minus the ten pages added that evening) and one "erased" new version. Since CP/M erases by placing an E5H in the first byte of the file entry in the directory, I used the North Star monitor to replace the E5 with a 00H, modified the filename to prevent confusion with the old version, and used the DOS WR command to write the directory back to diskette. Re-booting CP/M and running the word processor, I was further dismayed to find that the restored file included little of the lost text.

Not ready to give up, I returned to DOS and again examined the CP/M directory entries. I was particularly interested in the group numbers—since the word processor has crashed while trying to write to disk, perhaps it had not written the file entry back to disk. Perhaps some of the "unused" groups not included in the directory entries for the old or restored versions of the file contained parts of the new text.

To check out this possibility, I first had to be able to translate the group numbering system used by this version of CP/M into the corresponding North Star disk addresses, so that I could RD or WR particular groups at will. Since the groups were numbered 2-79 and I knew that my CP/M files began at North Star disk address 38 and could run through disk address 349, it was easy to deduce that if E were the group number, the corresponding North Star disk address would be (E-2)*4+38. With this knowledge, I was able to RD in the unused sectors and then examine the text with the monitor. A slow process to be sure, but within an hour I had found most of my missing text, determined the ordering of the groups (which was sequential with some jumps), and then "created" a new file entry into which I placed the series of groups. When I returned to CP/M and the word processor and tried to read this text, I found that it would only read a small piece of the reconstructed file. Returning to DOS and examining the end of this piece, I found a series of 1AH's—Control-Z's, the CP/M end-of-file mark. Zeroing these and rewriting the affected block to disk solved the problem. I went to bed reassured that the total loss of text had been reduced to one paragraph instead of ten pages!

Thomas B. Wiens, 2025 Westwood Terrace, Vienna, VA 22180.

Let a Program Do the Dirty Work

While the above procedure will work in a pinch, it is rather tedious—why not automate the whole process, taking advantage of what I now understand about CP/M diskette structure? This notion led to the North Star Basic program CP/MDP given in the listing. The program is written for the Lifeboat single-density version of CP/M 1.4, Release 5 single-density DOS and North Star Basic, and furthermore exploits the display control features of the IMSAI VIO-C memory-mapped video board. Obviously a bit of careful patching is required before the program will run correctly on other systems.

What does it do? First, it automatically provides a listing of a CP/M diskette directory, both "live" and "erased" entries, giving filenames (exactly as written on disk) and group numbers (translated into North Star disk addresses). Those files which are "live" are highlighted on the screen; if any groups of "erased" files have not

been overwritten by "live" files, the first appearances of those groups are also highlighted to indicate that they may contain salvageable information (though not necessarily the information you might expect !). After this listing is displayed on the screen, the user may also request a hardcopy printout.

Following the above, the directory information is used to display a "diskette map," which simply lists the contents of the diskette in order of disk address, indicating also where empty sectors occur.

A menu is then displayed, which allows the user to choose to:

1.REPEAT SECTOR MAPPING
2.MODIFY DIRECTORY
3.READ A SECTOR (HEX)
4.READ A SECTOR (ALPHA)
5.JUMP TO CP/M
6.JUMP TO DOS

The first option allows the user to either list the directory of another CP/M diskette or relist the previous one, perhaps to confirm that changes have been made as desired.

The second option requests the number of the directory entry to be changed (as previously listed), then allows the user to restore or erase the file, change the filename, change the number of records and/or change the group of numbers. One can, for example, take an exisiting erased entry, restore it to life, put in a new name, number of records (128 bytes per record; 8 records per group; and 128 records per entry) and set of disk addresses, thereby creating an entirely new file from groups which have not been committed to another live file (as I did when salvaging my text as described above). The program does not currently allow you to modify the extent numbers, although that could easily be added—they lie in the directory byte immediately following the filename.

The third option requests a North Star disk address and reads in 4 blocks of hex data. The user may view this by moving the cursor forward or backward on the screen with keyboard left and right arrows (or some other control keys, if statements 1040 and 1050 are modified). Typing in any two hex characters (0-9,A-F) replaces the pair under the cursor. Typing ESCAPE writes the modified data back to diskette; typing RETURN returns one to the menu. The most frequent use of this option is in patching .COM files. It can be used, for example, to examine and patch your BIOS I-O routines, the location of which on diskette is usually described in the implementation manual for any version of CP/M.

The fourth option also requests a disk address, reads in 4 blocks of ASCII text and displays it all on the screen (with ? marks for non-printable characters). This option can be used to search for missing or garbaged pieces of files. Unlike CP/M, it is not stopped by Control-Z's in the text, and so it can be used to pinpoint their locations. However, the third option must be used if the user wishes to modify the text (e.g., to remove garbage).

Options five and six provide jumps to a cold boot of CP/M, assuming that a CP/M diskette has been placed in drive 1, or to North Star DOS (the entry points used are given in the CALLs at statements 795-800).

What's the Gimmick?

North Star Basic, like CP/M, won't let you read or write from disk independent of the file structuring system. So how do we accomplish the above? The gimmick is to exploit North Star DOS' auto-start facility (see North Star *System Software Manual*, Rev. 2.1, pp. F-1 to F-2). To use CP/MDP, the DOS byte which initiates this facility must be set to zero (it is address 2030H; a FILL 8240,0 statement may be added to the beginning of the program to initialize this byte). One must also locate the DOS command input buffer (its location is stored at 2031-2032H; in the single-density version 5.0, the buffer begins at address 10109 decimal). A number of statements in the program stuff commands into this buffer, and may need modification for different version of DOS (statements 50, 980, and 1235-1245). Finally, we need an entry point into DOS which, if called, initiates an auto-start. This entry point is an "LDA 2030H" statement which, in my DOS, is found at address 8379 decimal; it may be located in other versions using the monitor command, SM 2000-2CFF 3A,30,20. Statements 70, 1090, and 1100 should be modified if the address is not the same. With these adjustments, when a read or write of the CP/M diskette is necssary, the program will stuff the appropriate command into the DOS buffer and call the auto-start entry point, which completes the command, leaving the user in DOS. To return to the program, it is necessary to type "JP 2A04" (2D04 in double-density versions) and "RUN n", where n is the desired CP/MDP entry point (all this is prompted by the program before it enters DOS.)

When the program reads blocks of text into memory, it stores them beginning at address 01. Since every RUN command re-initializes all variables, the program uses its own buffer at address 4091 decimal as temporary storage. Both these locations are arbitrary and can be changed if necessary.

As the program is written for my system, it exploits the power of the IMSAI VIO-C video board for an attractive display which highlights significant data (using character-by-character reverse video). The relevant video control codes, which will need to be replaced for other terminals/displays, are as follows:

CHR$(26) = protect/unprotect reverse video fields;

CHR$(22) = turn on/off individual character reverse video;

CHR$(16) = protect/unprotect reverse video fields;

CHR$(27)+" = "+CHR$(31+Y)+CHR$(31+X) positions the cursor at line Y and column X.

Also, for keyboards without positional keys (arrows), the ASCII numbers "9" and "8" in statements 1040 and 1050 will need to be replaced.

The formulas converting CP/M group numbers to North Star disk addresses are found in statements 235, 730, 925, and 935. For different versions of CP/M or DOS, these formulas may need to be modified. The first part of the article suggests how to deduce the required conversion formula.

In reading the CP/M directory, CP/MDP determines the number of file entries by examining the first byte of

each potential entry to see if it is a 0 or E5H; if it is not, the program assumes that the last entry has been reached. Should the directory itself get garbaged, this assumption may be incorrect, and the program may fail to read portions of the directory. In that case, use option 5 to jump to DOS and any of the monitors other than M0000 to examine the directory (DA 1,90 will do for the first block) and place a zero or E5H in the garbaged byte which is blocking the full directory read. After returning to DOS, use the WR command to write the directory back to diskette (8 blocks).

CP/MDP maps the diskette by setting up a vector X with 79 entries corresponding to groups 2-79 (the dimensions may need to be changed for other version of CP/M—see statements 140 and 720). If a group appears in "live" file, the entry number for that file is written as a positive number in X; if it appears in an "erased" file, it is written as a negative number. If the group appears in no directory entry, it is treated as "empty."

Saved Again

Ironically, I had written two-thirds of the text for this article when my word processor again crashed—this time because I had mistakenly directed the new text file to a crowded diskette. With CP/MDP available, no worry: I moved the injured diskette to disk 2, booted DOS on disk 1 and ran CP/MDP. The directory entry to the text file contained no group information, but I noted the disk addresses where there were blocks which were potentially "alive" but uncommitted to an unerased file. Using the fourth finding, I checked each of these, finding that they contained all but a small part of the lost text (of course, had the diskette been completely full, nothing could have been saved). Returning to option 2, I entered 8 addresses in the file directory entry, and computed and entered the number of records as 8x8 = 64. After having the directory written back to diskette, I moved the CP/M diskette back to drive 1, chose option 5 to boot CP/M, and was back in business.

```
**********************************************************************
*                                                                    *
*                LISTING: CP/MDP   AUTHOR:  T. WIENS                 *
*                                                                    *
**********************************************************************

 1REM THIS PROGRAM USES SOME VIDEO CONTROL CHARS. WHICH WILL NEED MODIFICA-
 2REM TION FOR OTHER SYSTEMS--CHECK ALL !CHR$() STATEMENTS.  DOS 5.0 SINGLE
 3REM DENSITY AND CP/M 1.4 SINGLE DENSITY (LIFEBOAT ASSOC. VERSION) IS ASSUMED.
 4REM FOR OTHER VERSIONS, CHECK CAREFULLY THE DOS LOCATIONS CALLED OR FILLED,
 5REM AND THE LAYOUT OF CP/M DISKETTES.  SOME KEY PARAMETERS\
 6REM 10109+ -- DOS COMMAND INPUT BUFFER,  8379 -- DOS AUTOSTART ENTRY POINT
 7REM 2A04H -- BASIC CONTINUE ENTRY POINT, 8232 -- DOS REENTRY POINT
 8REM 59648 -- COLD BOOT ENTRY, 4091+ -- BUFFER USER BY PROGRAM
 9REM 22146 -- TERMINAL CHAR.POINTER, 30 -- SECTOR WHERE CP/M DIR. BEGINS
10REM 38 -- FIRST SECTOR FOR CP/M FILES, 349 -- LAST SECTOR
11REM  2 -- FIRST CP/M GROUP NO., 79 -- LAST CP/M GROUP NUMBER
19 LINE#1,84\REM TO DEFEAT AUTOMATIC CARRIAGE RETURNS
20 !CHR$(26),"CP/M DISKETTE PROCESSOR by Thomas B. Wiens"
21 !"This program provides a means of reading & modifying CP/M diskettes,"
22 !"including editing CP/M directories to recover erased files or destroy"
23 !"garbage (even to piece together new files from pieces scattered around"
24 !"the diskette), reading individual blocks of text or hex files, patching"
25 !".COM files, and screen or hardcopy mapping of diskette structure."
26 !"    RAM starting at 01H is used as buffer space.  References to sector"
27 !"numbers are based on NS sector format, converted back and forth from"
28 !"CP/M's group numbering scheme.  For some commands, the program stuffs"
29 !"commands into DOS' input buffer and calls DOS to execute; to return"
30 !"to the program, follow the printed instructions EXACTLY."
33 !"    BEWARE OF MODIFYING THE CP/M DIRECTORY UNTIL YOU KNOW WHAT YOU"
34 !"ARE DOING--PRACTICE ON A DUPLICATE DISKETTE."
35 !"   NOTE: CP/M AND ITS PROGRAMS TAKES CONTROL-Z (1AH) AS AN END-OF-FILE"
36 !"MARK, AND WILL NOT READ PAST SAME, WHEREAS THIS PROGRAM WILL.  IF "
37 !"DESIRED, YOU CAN USE THE ROUTINES HEREIN TO ELIMINATE THE CONTROL-Z'S."
39 !\!"PRESS ANY CHARACTER TO CONTINUE..."\Y$=INCHAR$(0)
40 !CHR$(26),"INSERT YOUR CP/M DISKETTE IN DRIVE 2, THEN PRESS ANY KEY"
41 Y$=INCHAR$(0)
45 DATA 82,68,32,51,48,44,50,32,49,32,56,13
46 DIM Z(12)\REM CONTAINS ASCII CODES FOR DOS RD COMMAND
50 FORI=1TO12\READZ(I)\FILL 10108+I,Z(I)\NEXT
60 !\!"NOW, TYPE:  JP 2A04"
65 !"             RUN 100"
70 Z=CALL(8379)\REM JUMP TO DOS AUTOSTART
```

```
100 !CHR$(26),"*******  DIRECTORY LISTING FOLLOWS *******"
101!"ACTIVE FILES ARE LISTED IN REVERSE VIDEO; ERASED FILES IN NORMAL VIDEO"
102!"ERASED BUT PERHAPS SALVAGEABLE BLOCKS ARE ALSO IN REVERSE VIDEO!"
105B=0\L1=32\L2=64\P1=0
106 ERRSET 4000,E1,E2
110FORI=1TOL2*32STEP32
115 X=EXAM(I)
120IFX<>229ANDX<>0THEN EXIT130\REM 229=ERASED FILE EXTANT
122 X=EXAM(I+1)\IF X=229THENEXIT130
125NEXTI
130L2=(I-1)/32
140DIMD(L2),R(L2),N$(11*L2),M(L2,16),A$(11),X(79)\N$=""
150FORI=1TOL2
152K=(I-1)*32+1
155D(I)=EXAM(K)\IFD(I)=229THEND(I)=1\REM FILE EXTANT DEAD OR ALIVE
160FORJ=K+1TOK+11\N$=N$+CHR$(EXAM(J))\NEXTJ\REM FILE NAMES (11 CHARS)
165R(I)=EXAM(K+15)\REM NO. OF RECORDS IN EXTANT
167L=0
170FORJ=K+16TOK+31
172L=L+1
175X=EXAM(J)\IFX=0THENEXIT185
180 M(I,L)=X\REM CP/M BLOCK NO. (16 PER EXTANT)
182 NEXT J
185NEXTI
190 GOSUB 400
200!"NO. FILENAME   DISK SECTORS"
210FORI=1TOL2
215 IFD(I)=0THEN!CHR$(22),\REM REVERSE VIDEO IF "LIVE" ENTRY
216 IF P1=1 ANDD(I)=1THEN!#P1,CHR$(91),\REM BRACKET DEAD FILES
220A$=N$((I-1)*11+1,(I-1)*11+11)
225!#P1,%2I,I,%#3I," ",A$," ",
230FORJ=1TO16\IFM(I,J)=0THENEXIT240
232 IFM(I,J)<0THEN!CHR$(22),
235X=(ABS(M(I,J))-2)*4+38\!#P1,X," ",\IFM(I,J)<0THEN!CHR$(22),
238 NEXTJ
240 IF P1=1 ANDD(I)=0THEN!#1,CHR$(93),\REM RIGHT BRACKET
241IFD(I)=0THEN!CHR$(22),\!#P1\REM TURN OFF REV.VIDEO
242 IF I=20 OR I=40 THEN GOSUB 280
245NEXTI
247 !"PRESS ANY KEY FOR MEMORYMAP:"\Y$=INCHAR$(0)
250 GOTO 700
280 !"PRESS ANY KEY FOR NEXT DISPLAY; 'P' FOR HARD COPY:"
290Y$=INCHAR$(0)\IF Y$="P" THEN 305
300P1=0\RETURN
305P1=1\I=I-20
310!#P1,"NO. FILENAME   DISK SECTORS"\RETURN
400 FOR I=1TOL2
410 IF D(I)<>0THEN450
420 FOR J=1TO16\X(M(I,J))=I\NEXTJ
450 NEXTI
460 FOR I=1TOL2
470 IF D(I)<>1THEN550
480 FOR J=1TO16\IFX(M(I,J))<>0THEN500
490 X(M(I,J))=-I\M(I,J)=-M(I,J)\REM X MAPS DISKETTE, M NEG. IF BLOCK
500 NEXTJ\REM ERASED.  IF FILE ERASED BUT BLOCK NOT USED BY AN ACTIVE
550 NEXT I\REM FILE, TREAT THE BLOCK AS POTENTIALLY RECOVERABLE
560RETURN\REM THE 1ST TIME IT'S FOUND, THIS MAY BE INCORRECT!!
700 REM MEMORY MAP
701 REM USES PROTECTION FOR INVERSE VIDEO FIELDS TO FORMAT OUTPUT
702!CHR$(26),CHR$(16),CHR$(22),
705 !"**********       DISK MAP (BY NS SECTOR)   **********"\!
710 A$="                "
720 FOR I=2TO79\REM RANGE OF CP/M BLOCK NOS. IN VERS. 1.4
725 IF X(I)=X(I-1)THEN750
```

```
726 IF X(I)<>0THEN 727 ELSE A$="**EMPTY**  "\GOTO730
727 A$=N$((ABS(X(I))-1)*11+1,(ABS(X(I))-1)*11+11)
730 !(I-2)*4+38,"-",A$,"  !! ",CHR$(13),
750 NEXT I
760 !CHR$(22),CHR$(16)
770 !\!"PRESS ANY KEY TO CONTINUE:"\Y$=INCHAR$(0)
775 !CHR$(26)\!"CHOOSE ANY OF THE FOLLOWING:"
780 !"1 -- REPEAT SECTOR MAPPING"\!"2 -- MODIFY DIRECTORY"
785 !"3 -- READ A SECTOR (HEX)"\!"4 -- READ A SECTOR (ALPHA)"
787 !"5 -- JUMP TO CP/M"\!"6 -- JUMP TO DOS"
790 !"   YOUR CHOICE?"\Y$=INCHAR$(0)
795 X=VAL(Y$)\IF X=1THEN40\IF X=6 THENZ=CALL(8232)
800 IFX=5THENZ=CALL(59648)\IFX=2THEN850\IFX=3THEN1000\IFX=4THEN1100
810 GOTO 775
850 !CHR$(26),\INPUT"ENTRY NO.",N1
852 ERRSET 4001,E1,E2
855 K=(N1-1)*32+1\A$=N$((N1-1)*11+1,(N1-1)*11+11)
860 IF D(N1)=0 THEN865ELSE!"FILE ",A$," HAS BEEN ERASED. RESTORE IT (Y/N)?",
862 Y$=INCHAR$(0)\IF Y$<>"Y" THEN 870 ELSE FILL K,0\D(N1)=0\GOTO870
865 !"FILE ",A$," IS ACTIVE. KILL IT (Y/N)?",
867 Y$=INCHAR$(0)\IF Y$<>"Y" THEN 870 ELSE FILL K,229\D(N1)=1
870 !\!"(PUSH CR IF NO CHANGE)        OLD                NEW"
875 !"FILENAME (11 CHARS.):     ",A$,"     ",\INPUT A$
880 IF LEN(A$)<>11 THEN 900\N$((N1-1)*11+1,(N1-1)*11+11)=A$
885 FOR J=1TO11\FILLK+J,ASC(A$(J,J))\NEXTJ
900 !"RECORDS COUNT (128 PER EXTANT,"
901 !" OR 8 PER GROUP)          ",R(N1),"          ",\INPUT Y$
910 IF Y$=""THEN 920 ELSE R(N1)=VAL(Y$)
915 IF R(N1)<=128 THEN FILL K+15,R(N1)
920 !"FILE GROUPS (4 SECTORS EACH):"
925 FOR J=1 TO 16\X=(ABS(M(N1,J))-2)*4+38\IFX=30THENX=0
930 !"   # ",J,"                ",X,"                ",\INPUT Y$
935 IF Y$=""THEN945 ELSE X=(VAL(Y$)-38)/4+2
940 IF X<80 AND X>1 THEN FILL K+15+J,X ELSE 942\M(N1,J)=X\GOTO 945
942  FILL K+15+J,0 \M(N1,J)=0
945 NEXT J
950 !\!"WRITE DIRECT. TO DISK (Y/N) OR CR TO REVIEW ENTRY:"\Y$=INCHAR$(0)
955 IF Y$="Y" THEN 975\IF Y$="N" THEN 775 ELSE 855
970DATA 87,82,32,51,48,44,50,32,49,32,56,13
975 RESTORE 970
980FORI=1TO7\READ Z(I)\FILL 10108+I,Z(I)\NEXT
990 GOTO 60
1000 GOSUB 1200\!"          RUN 1010"\Z=CALL(8379)
1010 !CHR$(26),"USE LEFT OR RIGHT ARROWS TO VIEW OR POSITION CURSOR"
1015 !"TYPE TWO HEX CHARS. TO REPLACE THOSE UNDER CURSOR; TYPE AN "
1020 !"ESCAPE TO WRITE MODIFIED FILE BACK TO DISK; CR TO RETURN TO"
1025 !"MENU."\!CHR$(22),\I1=1\Y$="  "\Y1=36\X1=32
1030 !CHR$(27)+"="+CHR$(Y1)+CHR$(X1),\REM PLACE CURSOR ON SCREEN
1035 Y$(1,1)=INCHAR$(0)\X=EXAM(I1)\FILL 22146,0\X2=ASC(Y$)
1040 IF X2<>9THEN1050ELSE!FNH$(X),\I1=I1+1\X1=X1+3
1045 IF X1<=104THEN1030ELSEY1=Y1+1\X1=32\GOTO 1030
1050 IF X2<>8THEN1060ELSEI1=I1-1\X1=X1-3
1055 IFX1>=32THEN1030ELSEX1=104\Y1=Y1-1
1056 IFY1>35THEN1030ELSEY1=36\X1=32\I1=I1+1\GOTO1030
1060 IF X2<>13 THEN 1065\!CHR$(22)\GOTO 775
1065 IF X2=27THEN1080ELSE IFX2<48ORX2>70THEN1035
1070  Y$(2,2)=INCHAR$(0)\X2=FND(Y$)\IFX2>=0THENFILL I1,X2
1075 !Y$,\I1=I1+1\X1=X1+3\GOTO1045
1077DATA 87,82,32,44,50,32,49,32,52,13
1080RESTORE 1077\!CHR$(22)
1082 B$=""\X=EXAM(4090)\FORI=1TOX\B$=B$+CHR$(EXAM(4090+I))\NEXT
1084 !"WRITING SECTOR ",B$
1085 FORI=1TO10\READ Z(I)\NEXT
1085 FORI=1TO10\READ Z(I)\NEXT
```

```
1090 GOSUB1235\!"              RUN 775"\Z=CALL(8379)
1100 GOSUB 1200\!"           RUN 1110"\Z=CALL(8379)
1110 !CHR$(26),"SECTOR READS:"\!
1120 FORI=1TO1024\X=EXAM(I)
1130IF(X>31AND X<128)OR(X>9ANDX<14)THEN!CHR$(X), ELSE!"?",\NEXT
1140GOTO770\REM NON-ALPHA CHARS. PRINT AS '?'
1200 INPUT "SECTOR NO. FOR 4-SECTOR READ? ",X
1210DATA 82,68,32,44,50,32,49,32,52,13
1220 RESTORE 1210
1230 FOR I=1TO10\READ Z(I)\NEXT\B$=STR$(X)
1235 FOR I=1TO3\FILL 10108+I,Z(I)\NEXT\FILL 4090,LEN(B$)
1240 FOR I=1TOLEN(B$)\X=ASC(B$(I,LEN(B$)))\Z=10111+I
1243 FILL Z,X\FILL 4090+I,X\NEXT
1245 FOR I=4TO10\FILL Z+I-3,Z(I)\NEXT
1250 !\!"NOW, TYPE:   JP 2A04"
1260RETURN
1600 DEF FND(H$)\REM HEX/DEC CONVERTER FROM NS MANUAL
1620IFH$=""THEN1675\T=0
1630FORE=LEN(H$)TO1STEP-1\C=ASC(H$(E,E))
1640IF(C<ASC("0"))OR(C>ASC("F"))THENEXIT1675
16"0IF(C>=ASC("0"))AND(C<=ASC("9"))THENC=C-48
1!  IF(C>=ASC("A"))AND(C<=ASC("F"))THENC=C-55
1!  5IF(C>ASC("9"))AND(C<ASC("A"))THENEXIT1675
1 60T=T+C*(16^(LEN(H$)-E))
 665NEXT
1670RETURNT
1675RETURN-1
1680FNEND
1700DEF FNH$(D)
1730H1$=""
1740 FORI=1TO0STEP-1
1745D2=INT(D/(16^I))
1750 IF D2>=10 THEN H1$=H1$+CHR$(ASC("A")+D2-10)
1755IFD2<10THENH1$=H1$+CHR$(ASC("0")+D2)
1760D=D-(D2*(16^I))
1765NEXT
1770RETURNH1$
1775FNEND
4000 IF E1=106 THEN 150\REM TO PREVENT REDIMENSIONING ERROR
4001 IF E1<>85  STOP ELSE !"MUST RE-READ DIRECTORY!"\GOTO40
```

DOS/BIOS Directory and File Conversion in North Star UCSD Pascal — Part 1

Chris Young

The UCSD Pascal programming system is famous because it has been implemented on a wide variety of computers. One of those implementations is for the popular North Star Horizon computer and its MDS-A mini-disk system for S-100 computers. Programs in North Star Basic, although not compatible with most Basics, have become widely available through articles published in major computing magazines. With the introduction of UCSD Pascal for North Star, users of North Star products have an even more powerful program development system. Because of the extreme portability of UCSD Pascal, even more software will be available. North Star users now are also relieved of the task of converting software from one language to another as required for programs written in a Basic which is not compatible with North Star Basic.

There is one conversion problem still remaining for users of North Star Pascal. How does one access data used with their old North Star operating system? Such data might consist of data bases, numerical tables, text files, or "Tiny Pascal" programs which the user wishes to run in UCSD Pascal.

Because the UCSD Pascal is designed to be portable, it uses standardized file and disk directory formats. These formats, as one might expect, are not directly compatible with North Star DOS (Disk Operating System). An entire new set of disk software called BIOS (Basic I/O System) is provided by North Star. The first part of this article deals with a procedure to convert directories from DOS format to UCSD Pascal's BIOS format and back again automatically. Three Pascal programs to aid in conversion are discussed. DOSTOBIOS converts a North Star Basic/DOS format disk into a Pascal/BIOS readable disk. BIOSTODOS reads a BIOS directory and creates a North Star DOS directory on the disk. DOSCAT gives a catalog listing of a DOS directory from Pascal. The second part of this article deals with the conversion of the files' contents once the user has gained access to them through the converted directory.

These methods work for North Star UCSD Pascal I.5, Version 1, DQ-Release 2 and 3. The procedure assumes you are using Release 5.1 DQ DOS and one or two double density drives. The user must adapt this method to the system he has available. This will eliminate the need for such cumbersome phrases such as "except in single density" or "track 0 through 69 in double sided" in this discussion. Some of the procedures described will not work with earlier versions or other configurations of this system. I did not have access to other configurations and cannot speak with any authority about them.

Chris Young, 3119 Cossell Dr., Indianapolis, IN 46224.

Part I: Directory Conversion

Disk Sector Allocations

North Star software and data reside on tracks 0 through 34 of a ten sector-per-track mini-disk. Each double density sector contains 512 bytes of information. North Star allocates sectors 0 through 3 for the DOS directory (see Table 1). If the disk is to be used as a bootstrap disk, sectors 4 through 9 must contain a boot program. Otherwise this area is used for data. Pascal usually does not access this area because UCSD Pascal has its own plans for the first ten sectors.

In order to allow ease of customizing I/O routines, North Star has decreed that track 0 (sectors 0-9) be reserved for their normal DOS purposes. BIOS is tricked into thinking that physical tracks 1 through 34 are really tracks 0 through 33. Special "undocumented" procedures are required to make physical track 0 accessible to BIOS. DOS always has access to the entire disk. Note that BIOS does not maintain the North Star directory nor does DOS and its associated utilities maintain BIOS directories.

BIOS uses logical track 0 (physical track 1) for its directories and other purposes. To allow BIOS to access the DOS directory, we must move the data on tracks 0

through 32 down to tracks 2 through 34 of the same or another disk. Why not 1 through 34? Because physical track 1, although accessible to BIOS, must be kept "clean" so that the newly created BIOS directories do not destroy any DOS data. This means that any valuable data on tracks 33 and 34 must be moved to other disks before the conversion can continue.

Table 1. *Disk Sector Allocations.*

		DOS/BASIC		BIOS/PASCAL	
Physical Trk	Sec.	Logical Blk.	Use	Logical Blk.	Use
0	0	0,1			
	1	2,3	Directory	NOT	
	2	4,5			
	3	6,7		NORMALLY	
	4	8,9			
	5	10,11	Boot	ACCESSIBLE	
	6	12,13	program		
	7	14,15	or Data	BY PASCAL	
	8	16,17	areas		
	9	18,19			
1	10	20,21		0	NOT
	11	22,23	Data	1	USED
	12	24,25	areas	2	
	13	26,27		3	Primary
	14	28,29		4	directory
	15	30,31		5	
	16	32,33		6	
	17	34,35		7	Backup
	18	36,37		8	directory
	19	38,39		9	
2	20	40,41		10	
	21	42,43		11	New DOS dir-
	22	44,45		12	ectory loc.
	23	46,47		13	
	24	48,49		14	
	25	50,51		15	New DOS data
	26	52,53		16	areas
	27	54,55		17	
	:	:	:	:	:
	:	:	:	:	:
32	328	656,657		318	
	329	658,659		319	
33	330	660,661		320	
	331	662,663	Not moved	321	
	:	:	by DOSTO-	:	:
	:	:	BIOS, Must	:	:
34	340	696,697	be empty.	338	
	349	698,699		339	

Using The DOSTOBIOS Program

Listing 1 is a Pascal program called DOSTOBIOS that is used to create a Pascal/BIOS directory on a Basic/DOS disk. Before processing, some re-arranging of files should be done. All files in tracks 33 and 34 should be moved to other disks. Note also that BIOS only allows 77 files on a disk volume. If more exist, they must be copied to a separate disk, and deleted before conversion. The disk to be converted (which we will call the "SOURCE" disk) should be COmpacted using the DOS CO utility. Note that CO requires that no files overlap. This is also a requirement of the Pascal conversion process. Another disk called "DEST" should be initialized using the DOS "IN" command. DEST may be the same disk as SOURCE. If so, do not use the "IN" command. It is not advised to have SOURCE and DEST the same. If something goes wrong, you may ruin the DOS directory and leave the disk in an indeterminant state.

To copy the data from the SOURCE (i.e. DOS) disk to DEST (i.e. BIOS) disk, eX(ecute the program DOTOBIOS. Enter the unit numbers of SOURCE and DEST. The data is moved, the DOS directory is sorted, and the new BIOS directory is created. The sorted DOS directory is rewritten with corrections made for the new data locations.

At this point, the files may be read by Pascal programs as files which have a type of "Datafile." They are most easily accessed using the BLOCKREAD Pascal intrinsic with the files RESET as untyped files. The various F(iler functions such as T(ransfer, R(emove, K(runch, and C(hange may be used on the files, and other files may now be safely written on the disk. The problem of converting data within the files is highly application dependent. In Part II of this article we will discuss some Pascal PROCEDUREs and FUNCTIONs to ease the actual data conversions. We will discuss some of the internal workings of DOSTOBIOS.

Accessing Inaccessible Blocks

The key to accessing track zero from BIOS is an "undocumented" feature of the UCSD Pascal intrinsics UNITREAD and UNITWRITE. By using special parameters to these routines, the one track offset (ten blocks) which maps logical tracks into physical tracks is eliminated. Normally, a call to UNITREAD would look like this:

```
UNITREAD(UNITNUM,BUFFER,NUMBYTES,BLOCKNUM,TRANSCODE);
```

UNITNUM is the integer unit number to be read. BUFFER is a packed array of char. NUMBYTES is the integer number of bytes to be transferred. BLOCKNUM is the integer number of the block to be read. TRANSCODE is an integer transfer code which usually selects synchronous or asynchronous transfer. The special parameters look like this:

```
UNITREAD(UNITNUM,BUFFER,0,BLOCKNUM,2);
```

By setting NUMBYTES to zero and TRANSCODE to 2, the procedure will transfer 512 bytes from the physical block pointed to by the value in BLOCKNUM. According to a staff technical consultant at North Star, this works for Pascal-DQ Release 2 and 3. Tests of the DOSTOBIOS programs have verified that this works for the DQ version. This does not work for Pascal-S (single density version) since earlier versions allow negative block numbers for BLOCKNUM. Current versions do not allow negative BLOCKNUM.

The procedure MOVIT in DOSTOBIOS uses the TRANSCODE of 2 to copy tracks 0 through 32 down to 2 through 34. The tracks are copied last to first so that SOURCE and DEST can be the same disk.

BIOS Directories

BIOS directories as defined in these programs have two types of entries: HEAD records and FYLE records. A directory contains one HEAD record and up to 77 FYLE records. The HEAD describes the entire volume while the FYLEs describe each file.

HEAD has seven fields. PSTRTBLK is the integer starting block number of the directory file. This is always zero for HEAD records. PNEXT is the integer starting block number of the next file (i.e. last block + 1). PNEXT - PSTRTBLK = the length of the file in 512 byte blocks. For HEAD entries (with backup directory) PNEXT is 10. PTYP is the type of entry which is zero for HEAD entries. PFILNAM is the volume name. It is a "string[7]" or a "packed arrary[0..7]" with PFILNAM[0] as a length byte.

FILINVOL is the integer number of FILes IN the VOLume. DATEINIT is the date of the volume was initialized for data disks, and is the current date for system disks. Dates are of the following form:

```
type DATES=packed record

            MON:0..11;

            DAY:0..31;

            YEAR:0..99;

           end;
```

For our purposes, DATES can be declared as integers. TIME is the time of last access in most UCSD Pascal implementations. Because North Star has not implemented a real-time clock, this field should be ignored. The remaining space in HEAD records is unused.

The other kind of entry is called FYLE (because FILE is a reserved word, it could not be used). In FYLE entries, PSTRTBLK, PNEXT, and PTYP are similar to HEAD entries. They are the start block, next start block, and file type of each files. See Table 2 for file types. PFILNAM is the file name which is a "string[15]" or a "packed array[0..15] of char" with PFILNAM[0] as a length byte. LASTBYTES is the integer number of bytes in the last block which is always 512 in our application. DATE is the creation date of the file.

Table 2. *UCSD Pascal File Types.*

Code #	Type
1	Bad disk
2	Codefile
3	Infofile
4	Textfile
5	Datafile
6	Graffile
7	Fotofile
8 up	ILLEGAL

Unlike DOS, there are no blank entries in a BIOS directory. FILINVOL tells exactly the number of files, and likewise the number of FYLE entries in the directory. FYLE entries in BIOS always occur in the same order as the files actually reside on disk. For this reason, the DOS entries are sorted in increasing order with PSTRTBLK as the key. If the value of PASDIR[I+1].PNEXT is not equal to PASDIR[I].PSTRTBLK for some value "I" then there exits a blank space on the disk with a length which is the difference of the two.

The variable definitions for PASENTRY, PASDIRECT, DOSENTRY, and DOSDIRECT use an advanced feature of Pascal which is called "variant record definitions." It allows us to redefine a buffer with several different record formats similar to a "REDEFINES" clause in COBOL. For example, PASDIRECT is a buffer 2048 bytes long. We may refer to it as PASBFR which is "packed array[0..2047] of char," or as PASDIR which is "packed array[0..77] of PASENTRY." Each PASENTRY is one directory entry. Further, we define a PASENTRY as one of two types of records either HEAD or FYLE. To access, for example, the date the volume was initialized use:

```
PASDIRECT.PASDIR[0].HEAD.DATEINIT
```

That is: the variable PASDIRECT, as the array PASDIR, element zero, with HEAD record format, in the DATEINIT field. As another example, the type of the tenth file is:

```
PASDIRECT.PASDIR[10].FYLE.PTYP
```

That is: the variable PASDIRECT, array PASDIR, element 10 FYLE record format, and PTYP field.

The Pascal statement "with" allows us to narrow down the part of a record that we are working with. For example:

```
begin

 ...

  DOSDIRECT.DOSDIR[I].DSTRTBLK:=DOSDIRECT.DOSDIR[I].
  DSTRTBLK+20;

 ...

end;
```

can be replaced by:

```
with DOSDIRECT.DOSDIR[I] do

 begin

  ...

   DSTRTBLK:=DSTRTBLK+20;

  ...

 end;
```

The reason that such powerful constructs as variant records are required is that the UNITREAD and UNITWRITE intrinsics *must* operate on packed arrays of chars. We could pick out sections of the buffers and process them character by character as one might do in Fortran or Basic. However, it is easier to "overlay" arrays of records and access them in a more readable form. For details on use of variant record specifications and the Pascal "with" statement, see *Pascal Users Manual and Report* by Jensen and Wirth, or the UCSD Pascal manual.

After all entries have been converted and copied into the Pascal buffer, the buffer is written into blocks 2 through 5. Then it is written into 6 through 9. These areas are the primary and duplicate directories respectively. Any old BIOS directories on DEST are overwritten. The DOS directory is updated to reflect the new locations of the files. It is written to DEST. If SOURCE is different from DEST, then the directory on SOURCE is not touched. If SOURCE = DEST, the original DOS directory is overwritten.

The process may be verified by entering the F(iler and taking an E(xtended directory listing and comparing it to a DOS directory listing. All files should have starting block numbers which are ten greater than they had in DOS. File lengths are listed in 512 byte blocks so they will appear as half their original DOS values. File names are of the form < filenam >.DOS where < filenam > is the original DOS name.

Using the BIOSTODOS Program

Occasionally the user may wish to create a DOS directory from a Pascal/BIOS directory. The program BIOSTODOS assists in this process. Because BIOS file names can be up to fifteen characters long, while DOS

names are limited to eight, the user must supply DOS names for the files. All other aspects of the conversion are automatic.

To create a DOS directory eX(ecute BIOSTODOS. (See Listing 2.) The user is first prompted by "Unit #" which requests the unit containing the disk to be converted. Unit must be a number; volume names or abbreviations such as "#" or ":" are not allowed. The user is then prompted by:

```
"Type DOS directory file name:"
```

This requests the name of a file which occupies track 0. Usually this file indentifies the name or ID number of the disk. Next, a file containing the Pascal/BIOS directory is created. The user is prompted for the file name by:

```
"Type Pascal directory file name:"
```

Each file name from the BIOS directory is then printed. After each, type a new DOS name for the file. All file names must be one to eight characters long, and must be unique. After all files have been given names, the user is asked:

```
"Update DOS directory?"
```

A response of "Y" or "y" will write the DOS directory and destroy any previous DOS directory. Any other response aborts the program and leaves everything intact. The BIOS directory is always left intact.

BIOSTODOS uses the same variables and record descriptions as DOSTOBIOS. The special parameters to UNITWRITE are used to access the DOS disk area. A function called GETNAME is used to input the new DOS names and check them for uniqueness. GETNAME returns the integer length of the name. This also doubles as an error flag. In the event of an error in GETNAME, a value of nine is returned. GETNAME has one variable parameter NAME. NAME is a "packed array[0..7]of character." Because the syntax for < parameter list > requires a < type identifier >, a new type PA07OC is defined as "Packed Array [0..7] Of Char." BIOSTODOS creates all file entries as double density type 0 files. The DOS start block number is ten more than in BIOS, and the length is computed by:

```
DFILLEN:=FNEXT - FSTRTBLK;
```

Note that no files are actually moved. The directory is created, written, and that's all.

Using The DOSCAT Program

Another useful program can be found in Listing 3. DOSCAT produces a standard DOS directory catalog listing from the USCD operating system. To use DOSCAT, eX(ecute) DOSCAT, and type the unit number the disk is in. As in BIOSTODOS, this must be a number. NUMLINES determines the number of files that will fit on a CRT screen. After the screen is full, type space to continue.

The internal workings of DOSCAT are rather straightforward. The procedure WRITEHEX is of interest. It takes an integer mod 256 and outputs it as a two "digit" hexadecimal number. WRITEHEX is used to ouput start addresses for DOS type 1 machine code files.

North Star Pascal Release 1,2 and 3

I recently received a copy of the UPGRADE: diskette for North Star Pascal. One of the programs on North Star's UPGRADE diskette creates a North Star directory reflecting the contents of the Pascal directory. This is the same function as Chris Young's program presented in Listing 2. Since the source code of the North Star program is available on the UPGRADE diskette, I compared Chris's program with the North Star program. I was surprised to find a significant difference in the method used to access the system track (track 0) on the Pascal diskette. The DOSDIR program on the UPGRADE: diskette accesses the system track by using a negative value for the BLOCKNUM parameter in the UNITREAD and UNITWRITE commands. In Chris's program, the physical block number of the desired block is used for the BLOCKNUM parameter, along with NUMBYTES=0 and TRANSCODE=2. Chris refers to this technique as a undocumented method obtained from a staff technician at North Star. I also noticed that Chris documented his Pascal as version 1, release 2.

I contacted North Star to find out what had been done to Pascal version 1, release 1 (my release) to allow Chris's version to work. Initially, all I got was confusion. I contacted Dave Gersen after he returned from vacation and he was able to supply a somewhat rational explanation. Dave Gersen promised to send me a memo he had prepared to clear up the confusion over the North Star Pascal product. Briefly, here is Dave's summary:

There have been three releases of version 1 (UCSD Pascal version 1.5). Release 1 was first sold in early 1979 (before Dave's time at North Star) and was very cheap, costing less than $100 for two diskettes and the manual. The UPGRADE: diskette was released in December of 1979 to provide two programs that were omitted from release 1, LIBRARIAN.CODE and LIBMAP.CODE. The additional utility programs on the UPGRADE: diskette (DOSDIR.CODE, IN.CODE and CD.CODE) were mistakenly included. In early 1980, North Star offered version 1, release 2 for about $199. North Star felt release 1 was underpriced, and the royalties North Star had to pay did not allow for much profit. Here is where the story gets a bit fuzzy.

During the summer and fall of 1979, North Star hired a contract programmer to review the Pascal BIOS and 'improve' it. This person made changes that North Star was unaware of; namely the handling of the BLOCKNUM parameter in the UNITREAD and UNITWRITE intrinsics. Release 2 was followed quickly by the current release 3. Release 3 is very similar (only four bytes changed) and fixes a bug with quad access.

Listing 1.

```
    (* Program to write a U.C.S.D. Pascal/BIOS directory *)
    (*  on a DOS disk, and to move data to areas easily  *)
    (*  accessible to Pascal.                            *)
          (* Written Aug. '80                           *)
          (*      by Chris Young                        *)
          (*         3119 Cossell Drive                 *)
          (*         Indianapolis IN 46224              *)
          (*            (317)-291-5376                  *)
program DOSTOBIOS;
 const DEFTYPE=5;  (* Pascal file type "Datafile" *)
       MAXFILS=77; (* Maximum number of PASENTPYs *)
       PHYSICAL=2; (* Transfer code for physical blocks *)
       LOGICAL=0;  (* Transfer code for logical blocks  *)
 type DOSENTRY=packed record
                DFILNAM:packed array[0..7] of char;
                DSTRTBLK:integer;
                DFILLEN:integer;
                DTYP:packed array[0..3] of char;
               end;
      PASPNTRY=packed record
                case integer of
                 0:(HEAD:packed record
                          PSTRTBLK:integer; PNEXT:integer;
                          PTYP:integer;
                          PFILNAM:packed array[0..7]of char;
                          BLKINVOL:integer;
                          FILINVOL:integer;
                          TIMF:integer;
                          DATEINIT:integer;
                         end);
                 1:(FYLE:packed record
                          PSTRTBLK:integer; PNEXT:integer;
                          PTYP:integer;
                          PFILNAM:packed array[0..15]of char;
                          LASTBYTES:integer; DATE:integer;
                         end);
               end;(*PASENTRY*)
var SOR,DEST,DCOUNT,PCOUNT,NAMLEN,DINDX,I:integer;
    CH:char;
    DOSTAG:packed array[0..3]of char;
    DOSDIRECT:packed record
         case integer of
          0:(DOSDIR:packed array[0..127]of DOSENTRY);
          1:(DOSBFR:packed array[0..2047] of char);
         end;
    PASDIRECT:packed record
         case integer of
          0:(PASDIR:packed array[0..MAXFILS]of PASENTRY);
          1:(PASBFR:packed array[0..2047] of char);
         end;
     (* Function which sorts DOSDIR in increasing order with *)
     (*  DSTRTBLK as a key.  Returns with integer count of   *)
     (*  the number of files in DOSDIR.                      *)
  function SORT:integer;
   var FILCOUNT,I,J,K:integer;
       DNTRY:DOSENTRY;
  begin
   WRITE('Sorting directory');
   with DOSDIRECT do
    begin
     FILCOUNT:=0;
       (* Eliminate blank and zero length files.  *)
     for I:=0 to 127 do
      if (DOSDIR[I].DFILNAM[0]=' ') or (DOSDIR[I].DFILLEN=0)
       then DOSDIR[I].DSTRTBLK:=9999;
     for I:=0 to 127 do
      begin
       K:=I;
       for J:=I to 127 do
        if DOSDIR[J].DSTRTBLK < DOSDIR[K].DSTRTBLK then K:=J;
       DNTRY:=DOSDIR[I];
       DOSDIR[I]:=DOSDIR[K];
       DOSDIR[K]:=DNTRY;
       if DOSDIR[I].DSTRTBLK <> 9999
        then FILCOUNT:=FILCOUNT+1;
       WRITE('.');
       if (I mod 32)=31 then WRITELN;
      end;(*for I *)
    end;(*with DOSDIRECT*)
    WRITELN;WRITELN('Sort complete.');
    SORT:=FILCOUNT;
  end;(*SORT*)
 (* Procedure to move all data from tracks 0 through 32  *)
 (*  to tracks 2 through 34.  Reads unit numbers for     *)
 (*  source and destination disks. Uses special transfer *)
 (*  code in UNITREAD and UNITWRITE so they access       *)
 (*  physical blocks instead of logical blocks.          *)
 procedure MOVIT;
  var TRACK:packed array[0..5120] of char;
      I,J:integer;
  begin
   repeat WRITE('Source unit #'); READLN(SOR);
    until (SOR=4) or (SOR=5);
   repeat WRITE('Destination unit #'); READLN(DEST);
    until (DEST=4) or (DEST=5);
   WRITE('Insert disks.  Are you ready? (Y/N):');
   READLN(CH);
   if (CH<>'Y') and (CH<>'y') then begin
                                    WRITELN('Exiting...');
                                    EXIT(DOSTOBIOS);
                                   end;
   WRITELN('Moving data');
   for I:=32 downto 0 do
    begin
     for J:=0 to 9 do
      UNITREAD(SOR,TRACK[512*J],0,I*10+J,PHYSICAL);
     for J:=0 to 9 do
      UNITWRITE(DEST,TRACK[512*J],0,I*10+J+20,PHYSICAL);
     WRITE('.');
    end;
   WRITELN; WRITELN('Data moved');
```

```
 end;(*MOVIT*)
begin
 WRITELN('DOSTOBIOS version 2.0');
 MOVIT;
 UNITREAD(DEST,DOSDIRECT.DOSBFR,2048,10,LOGICAL);
 UNITREAD(DEST,PASDIRECT.PASBFR,2048,2,LOGICAL);
 DCOUNT:=SORT;
 if DCOUNT > MAXFILS then begin
                           WRITELN('TOO MANY FILES');
                           WRITE('Hit <sp>');READ(CH);
                           EXIT(DOSTOBIOS)
                          end;
 DOSTAG:='.DOS';
 PCOUNT:=1;
 WRITE('Copying directory');
 for DINDX:=0 to DCOUNT do
  with PASDIRECT.PASDIR[PCOUNT].FYLE,
       DOSDIRECT.DOSDIR[DINDX] do
   begin
      (* Output some dots *)
    if (DINDX mod 50)=49 then WRITELN('.')
                         else WRITE('.');
      (* Skip blank and zero length records *)
    if (DFILNAM[0]<>' ') and (DFILLFN<>0) then
     begin
      NAMLEN:=SCAN(8,=' ',DFILNAM);
      MOVELEFT(*to right*)((*source*) DFILNAM[0], (*to*)
                           (*destination*) PFILNAM[1], (*for a *)
                           (*length of *) NAMLEN);
      MOVELEFT(*to right*)((*source*) DOSTAG[0], (*to*)
                           (*destination*) PFILNAM[NAMLEN+1],
                           (*for length of*) 4);
      NAMLEN:=NAMLEN+4;
      PFILNAM[0]:=CHR(NAMLEN);
      PSTRTBLK:=DSTRTBLK+10;
      PNEXT:=PSTRTBLK+DFILLEN;
      PTYP:=DEFTYPE;
      LASTBYTES:=512;
      DATE:=0;
      PCOUNT:=PCOUNT+1;
      (* Update DOS start block *)
      DSTRTBLK:=DSTRTBLK+20;
     end;(*if DFILNAM<>' ' and DFILLFN<>0*)
   end;(*for DINDX; with PASDIRECT,DOSDIRECT*)
      (* Create BIOS directory header record *)
      with PASDIRECT.PASDIR[0].HEAD do
       begin
        PSTRTBLK:=0;
        PNEXT:=10;
        PTYP:=0;
        MOVELEFT(*to right*)((*source*) DOSTAG[1], (*to*)
                             (*destination*) PFILNAM[1],
                             (*for length of*) 3);
        PFILNAM[0]:=CHR(3);
        BLKINVOL:=340;
        FILINVOL:=PCOUNT-1;
        DATEINIT:=0; TIME:=0;
       end;
      (* Create new directory files in DOS directory *)
      with DOSDIRECT.DOSDIR[0] do
       begin
        DFILNAM:='DOSDIR  ';
        DSTRTBLK:=0;
        DFILLEN:=10;
        DTYP[0]:=CHR(128);
       end;
      with DOSDIRECT.DOSDIR[1] do
       begin
        DFILNAM:='PASDIR  ';
        DSTRTBLK:=10;
        DFILLEN:=10;
        DTYP[0]:=CHR(128);
       end;
      (* Blank out unused file entries *)
      for DINDX:=DCOUNT+1 to 127 do
       with DOSDIRECT.DOSDIR[DINDX] do DFILNAM[0]:=' ';
      UNITWRITE(DEST,PASDIRECT.PASBFR,2048,2,0); (* Primary dir. *)
      UNITWRITE(DEST,PASDIRECT.PASBFR,2048,6,0); (* Back-up dir. *)
      (* Rewrite updated DOS directory *)
      for I:=0 to 3 do
       UNITWRITE(DEST,DOSDIRECT.DOSBFR[I*512],0,I,PHYSICAL);
end.
```

Listing 2.

```
                (* Program to write a DOS directory    *)
                (*    on a U.C.S.D. Pascal/BIOS disk. *)
                (*  Written Aug. '80                  *)
                (*      by Chris Young                *)
                (*         3119 Cossell Drive         *)
                (*         Indianapolis IN 46224      *)
                (*            (317)-291-5376          *)
(**)
program BIOSTODOS;
 const MAXFILS=77;  (* Maximum number of BIOS files  *)
       DOSTYPE=128; (* File type for new DOS entries *)
 type PA070C=packed array[0..7]of char; (* see func. GETNAME *)
      DOSENTRY=packed record
                 DFILNAM:packed array[0..7] of char;
                 DSTRTBLK:integer;
                 DFILLEN:integer;
                 DTYP:packed array[0..3] of char;
                end;
       PASENTRY=packed record
                  case integer of
                   0:(HEAD:packed record
                             PSTRTBLK:integer; PNEXT:integer;
                             PTYP:integer;
                             PFILNAM:packed array[0..7]of char;
                             BLKINVOL:integer;
                             FILINVOL:integer;
```

```
                        TIME:integer;
                        DATEINIT:integer;
                       end);
                  1:(FYLE:packed record
                        PSTRTBLK:integer; PNEXT:integer;
                        PTYP:integer;
                        PFILNAM:packed array[0..15]of char;
                        LASTBYTES:integer; DATE:integer;
                       end);
              end;(* PASENTRY*)
var UNITNUM,PLEN,DLEN,INDX,NUMPFILES,I:integer;
    CH:char;
    DOSDIRECT:packed record
         case integer of
          0:(DOSDIR:packed array[0..127]of DOSENTRY);
          1:(DOSBFR:packed array[0..2047] of char);
         end;
    PASDIRECT:packed record
         case integer of
          0:(PASDIR:packed array[0..MAXFILS]of PASENTRY);
          1:(PASBFR:packed array[0..2047] of char);
         end;
function GETNAME(var NAME:PA07OC):integer;
(* NOTE:the following header is illegal syntax.....
(* function GETNAME(var NAME:packed array[0..7]of char;):integer
(*                          <= type identifier expected
(* .....that is why there is a type PA07OC *)
 var SNAME:string[20];
     I,LEN:integer;
     MATCH:boolean;
 begin
  READLN(SNAME); LEN:=LENGTH(SNAME);
  if (LEN <= 8) and (LEN > 0)
   then begin
         FILLCHAR(NAME,8,' ');
         MOVELEFT(*to right*)((*source*) SNAME[1], (*to*)
                              (*destination*) NAME[0],
                              (*for length of*) LEN);
         MATCH:=false;
           (* INDX is the number of the file we are working *)
           (*  on.  No MATCH look-up is needed 1st pass.    *)
         if INDX > 0 then
          for I:=0 to INDX-1 do
           with DOSDIRECT.DOSDIR[I] do
             MATCH:=MATCH or (NAME=DFILNAM);
         if MATCH then
          begin
           LEN:=9;
           WRITELN('ERROR Name "',SNAME,'" already used');
          end;(*if MATCH*)
        end(*if LEN<=8 then*)
   else WRITELN('ERROR Name too long or short.');
  GETNAME:=LEN;
 end;(*GETNAME*)
```

```
begin
 WRITELN('BIOSTODOS version 2.0');
WRITE('Unit #'); READLN(UNITNUM);
UNITREAD(UNITNUM,PASDIRECT.PASBFR,2048,2,0);
FILLCHAR(DOSDIRECT.DOSBFR,2048,' ');
with PASDIRECT,DOSDIRECT do
 begin
  NUMPFILES:=PASDIR[0].HEAD.FILINVOL;
  with DOSDIR[0] do
   begin
    repeat
     begin WRITE('Type DOS directory file name:');
           INDX:=0; DLEN:=GETNAME(DFILNAM);
     end until DLEN <=8;
    DSTRTBLK:=0;
    DFILLEN:=10;
    DTYP[0]:=CHR(DOSTYPE);
   end;
   with DOSDIR[1] do
    begin
     repeat
      begin WRITE('Type Pascal directory file name:');
            INDX:=1; DLEN:=GETNAME(DFILNAM);
      end until DLEN <=8;
     DSTRTBLK:=10;
     DFILLEN:=10;
     DTYP[0]:=CHR(DOSTYPE);
    end;
   for INDX:=2 to NUMPFILES+1 do
    with PASDIR[INDX-1].FYLE,DOSDIR[INDX] do
     begin
      PLEN:=ORD(PFILNAM[0]);
      repeat
       begin
        for I:=1 to PLEN do WRITE(PFILNAM[I]);
        for I:=PLEN to 20 do WRITE(' ');
        WRITE(' New DOS name... "........"');
        for I:=0 to 8 do WRITE(CHR(8));
        DLEN:=GETNAME(DFILNAM);
       end until DLEN <=8;
      DSTRTBLK:=PSTRTBLK+10;
      DFILLEN:=PNEXT-PSTRTBLK;
      DTYP[0]:=CHR(DOSTYPE);
     end;(*for INDX with PASDIRECT,DOSDIRECT*)
  end;(*with PASDIRECT,DOSDIRECT*)
 WRITE('Update directory?'); READ(CH);
 if (CH<>'Y') and (CH<>'y') then begin
                                  WRITE('Exiting...');
                                  EXIT(BIOSTODOS);
                                 end;
 for I:=0 to 3 do
  UNITWRITE(UNITNUM,DOSDIRECT.DOSBFR[I*512],0,I,2);
end.(*BIOSTODOS*)
```

Listing 3.

```
            (* Program to list a DOS directory    *)
            (*    from U.C.S.D. Pascal.           *)
            (*  Written Aug. '80                  *)
            (*      by Chris Young                *)
            (*         3119 Cossell Drive         *)
            (*         Indianapolis IN 46224      *)
            (*            (317)-291-5376          *)
program DOSCAT;
 const NUMLINES=23; (* Number of lines on screen - 1 *)
       DIRSIZ=127;  (* Number of DOSENTRYs           *)
       PHYSICAL=2;  (* Transfer code for UNITREAD    *)
 type DOSENTRY=packed record
                DFILNAM:packed array[0..7] of char;
                DSTRTBLK:integer;
                DFILLEN:integer;
                DTYP:packed array[0..3] of char;
               end;
 var UNITNUM,INDX,I,TYP:integer;
     CH:char;
     DOSDIRECT:packed record
          case integer of
           0:(DOSDIR:packed array[0..127]of DOSENTRY);
           1:(DOSBFR:packed array[0..2047] of char);
          end;
 (* Procedure to write a hexidecimal VAL to OUTPUT *)
 (*  Takes VAL mod 256 and writes 2 hex "digits"   *)
 procedure WRITEHEX(VAL:integer);
  var HEXDIG:packed array[0..15] of char;
  begin
   VAL:=VAL mod 256;
   HEXDIG:='0123456789ABCDEF';
   WRITE(HEXDIG[VAL div 16]);
   WRITE(HEXDIG[VAL mod 16]);
  end;(*WRITEHEX*)

begin
 WRITELN('DOSCAT version 2.0');
 WRITE('Unit #'); READ(UNITNUM);
 for I:=0 to 3 do
  UNITREAD(UNITNUM,DOSDIRECT.DOSBFR[I*512],0,I,PHYSICAL);
 with DOSDIRECT do
  for INDX:=0 to DIRSIZ do
   with DOSDIR[INDX] do
    begin  (* Ignore blank entries *)
     if DFILNAM[0]<>' ' then
      begin
       WRITELN;
       WRITE(DFILNAM,DSTRTBLK:6);
       TYP:=ORD(DTYP[0]);
  if TYP>127  (* Test density bit *)
   then begin
         TYP:=TYP-128; WRITE(DFILLEN*2:6,' D');
        end
   else begin
         WRITE(DFILLEN:6,' S');
        end;
       WRITE(TYP:4);
       (* Test for type 1 machine code file *)
       if TYP=1 then begin WRITE(' ');
       (* High order byte *) WRITEHEX(ORD(DTYP[2]));
       (* Low order byte  *) WRITEHEX(ORD(DTYP[1])); end;
       (* Test for full screen *)
       if (INDX mod NUMLINES)=(NUMLINES-1) then
        begin
         WRITE((* Home cursor *) CHR(11),'Hit <sp> to continue');
         READ(KEYBOARD,CH);
         WRITELN((* Clear screen *) CHR(12));
        end;
      end;(*if DFILNAM[0]<>' '*)
    end;(*with DOSDIRECT; for INDX; with DOSDIR[INDX]*)
end.
```

DOS/BIOS Directory and File Conversion in North Star UCSD Pascal — Part 2

Chris Young

Part II—File Conversion

In this installment we will discuss how to convert the information within the files into standard Pascal data and file types. We will look at some Pascal procedures that facilitate the conversion of North Star data files into similar Pascal data files. This is accomplished by reading the North Star data into standard Pascal types of variables where further processing, including the output of the data to Pascal files, may be done. Three kinds of conversion are discussed: 1) North Star Basic text files into UCSD format text files. 2) North Star Basic string data into Pascal "packed array [..] of char" data. 3) North Star Basic BCD floating point data into Pascal type "real" data.

Text File Conversion

Why should Basic text files be of concern to Pascal users? Prior to the introduction of the new North Word word processing system and Pascal, users did not have a text editing system *specifically* designed for use with North Star systems. True, one can spend over a hundred dollars for CP/M and more for Electric Pencil, Wordstar, or others. However, users of small systems (and users who spent their entire budgets on large systems) may not be able to afford these powerful editors. Even North Word isn't cheap. Many of these users resort to using the Basic program entry editor for their word processing needs. This is accomplished by creating text files full of REM statements. In fact, as long as the user does not try to RUN the files, even the word REM is not necessary. The Basic editor, although limited, can help the user get by on a budget. Most often these files contain documentation of Basic programs, but many other word processing uses can be covered by the Basic editor. A user may have this or many other reasons why he wishes to access Basic text files from Pascal. As his pocketbook recovers, and his editing needs increase, the user may come to the conclusion that for a smaller expenditure for software than is needed for many text editors, he can have the entire UCSD system. This statement is made with the realization that perhaps a significant expenditure for hardware to upgrade to 48K may be necessary. The UCSD system includes a complete Pascal program development system consisting of a compiler, linker, file manager, a Z80 and an 8080 macro assembler, and a powerful screen-oriented text editor. The program described below allows users to access old data files, for whatever reason, for use in this versatile new system.

When a user types a Basic program source text into North Star Basic, the text is not stored exactly as it was entered. Basic recognizes keywords and commands, compacting them into one byte tokens in the range of 128 to 255. The feature not only saves space, but results in faster interpretation of code. When LIST commands are issued, a simple table look-up expands the "crunched" tokens back into their exact original form. However, this space saving is not limited only to Basic statements. Text in string literals and comments (REM statements) are also compacted into one byte tokens wherever possible.

I will now describe the program CVTBAS, found in Listing 4, which performs the function of re-expanding compressed keyword tokens into strings of characters. It creates a UCSD text file or outputs the text to any system device.

First I will examine the procedure GETCHAR which reads the original untyped file, one block at a time, and feeds the characters out one byte at a time. This deblocking routine is the basic tool used in all North Star to UCSD data conversions. GETCHAR communicates through five variables and an input file which must be declared globally (i.e. at the outermost program level). The file "DOS" is declared as an untyped file (i.e. "var DOS:file;"). LASTCHR is a boolean flag which enables/disables the transmission of characters out of GETCHAR. The flag must be initialized to false at the opening of the input file. After the last byte of the last block has been transmitted, GETCHAR sets LASTCHR to true. Any further calls to GETCHAR returns a null byte. North Star uses a value of one (ASCII character "SOH") as an end of file marker, however,

this is context dependent. GETCHAR cannot set LASTCHR true upon reaching a North Star end of file. The program calling GETCHAR can determine if the byte value of one is part of the data or an end of file mark, and it sets, LASTCHAR is necessary. The character itself is passed to the calling routines in two global variables: CH and CHINT. CH is of type "char" and CHINT is an "integer" whose value is ORD(CH). BUFR is a "packed array [0..511] of char" which is used as the input buffer. BFPTR is an "integer" which points into BUFR. Each time GETCHAR is called, BFPTR is incremented by one. When BFPTR reaches 512, the buffer is empty and a new block is read. To read the first block, the calling program should initialize BFPTR to 512, forcing a block to be read upon the first call to GETCHAR.

The procedure DOLABEL processes Basic labels which are in the form of line numbers. North Star Basic allows line numbers in the range of 1 through 65535 (64K-1). The line numbers are stored as 16-bit <un-signed> integers. Because UCSD Pascal limits integer variables to -32K to +32K, type "real" variables must be used in the calculations which output the line number. Line numbers appear at the beginning of each line and in certain Basic statements. All in-statement label references are preceded by a token code 154. When a code 154 is received at the beginning of each line, the LABL flag is set to true. This causes the invocation of DOLABEL.

DOLABEL uses five variables. LINENUM is of type "real" and is used to store the line number value. P, also type "real," holds a power of ten to be used to strip off the highest order digits one at a time. LEADZERO is a boolean flag, initialized to true upon entry to DOLABEL, which suppresses the output of leading zeros. LEADZERO is reset to false when the first non-zero digit is processed.

DOLABEL, as well as other routines in the system, requires a character to be in CH and CHINT upon entry to the routine. Likewise, all routines also call GETCHAR upon exit, thereby leaving the next character ready for the next section of processing. Upon entry, DOLABEL initializes LEADZERO and puts the low order byte of the line number in LINENUM. Then GETCHAR is called to get the high order byte. This byte is "shifted left" one byte by multiplying it by 256.0 (remember to use "real" arithmetic). Next add in the low order byte and save it in LINENUM. Because we do not want a decimal point output with the value, we do not use the standard WRITE procedure to output the real valued number. Instead, we strip off the high order digits, one at a time, truncate them to one digit integers, and output them to the output device. The loop runs from 4 down to 0 and the variable P is assigned a value of ten to the fourth down to ten to the zero power. This is used to compute what amounts to integer division and modulo calcuations on a real value. As each digit is stripped off, proceeding left to right, the LEADZERO flag is updated and tested. Non-leading zero digits are output to the output device and to the CONSOLE:. The function of the ISFILE flag will be discussed later.

The main program is initialized by two routines: INIT1 and INIT2. Two routines are needed because there is a limit on the size of procedures. The procedures initialize the table as "KEY:packed array [0..255]of string[10]." The array elements are all first set to a one character string which is the normal ASCII value for that character. Individual elements are then set to strings of characters which will replace the compacted tokens.

After INIT1 and INIT2 are called, the user is prompted to type the name of the Basic text file to be converted. The response is stored in INPNAME and the file is opened by RESET. The user is then prompted for the output file name. If he hits return, the length of OUTNAME is zero, and ISFILE is set false. If a name is entered, the ISFILE flag is set true, and OUTDEV is created as a text file by REWRITE. The text produced by this program is always sent to the CONSOLE: device. If ISFILE is true, the output is also sent to the file OUTDEV.

All lines begin with a length byte which we will skip over. Next, each line has a label. To process the first one, the LABL flag is set to true. A North Star EOF test is made and we enter a "while not LASTCHR do" loop. Another byte is obtained from GETCHAR because all routines require a character to be in CH and CHINT upon entry. IF LABL is true we call DOLABEL and loop back again. Otherwise, a keyword look-up is done by using CHINT as the index into KEY. The string stored in KEY[CHINT] is output. If CHINT is a carriage return (code 13), then we do an extra GETCHAR to ignore the length byte. This is another place where we test for CHINT=1, meaning a North Star EOF was read. Processing continues as above until LASTCHR is true.

Data File Conversion

Users are much more likely to have a need to convert data files from North Star to Pascal formats. These files may be data bases, tables, mailing lists or any one of a number of other types. The overall format of data files is very application dependent. Rather than trying to give samples of specific data conversions, we will present a general conversion program named CVTDATA (See Listing 5). CVTDATA reads a North Star file and outputs its contents to the CONSOLE: device. The program merely demonstrates the techniques needed. The user must adapt the routines and code sections needed to read the North Star data. Once the data is in Pascal variables, further processing may be done on the data, including output to a new file.

Data in North Star disk files is one of four types: an end of file marker one byte long, a floating point number which occupies five bytes in the standard eight digit version, short strings which occupy 1 to 256 bytes, and long strings which occupy more than 256 bytes of space, with a maximum length which is limited by available memory. CVTDATA reads the input file, determines which of these four types it has encountered, and passes the data to the CONSOLE: with a message telling what type of data was read.

String data is processed by the main program and will be covered later. Floating point data is processed in a separate routine and requires some background information before it may be discussed.

North Star floating point data consists of an eight digit binary coded decimal (BCD) normalized mantissa in four bytes, and an exponent in the fifth byte. The high order bit of the exponent byte contains the sign of the mantissa. The remaining seven bits are an excess 64 exponent

base 10. A zero exponent means a value of zero for the entire floating point number. UCSD real variables occupy four bytes. There is a 23-bit binary normalized mantissa with an implied leading 1 bit, a mantissa sign bit, and an 8 bit excess 128 exponent base 2. Zero exponents also denote zero value as in North Star.

The user of North Star/ USCD Pascal now has all of the necessary tools to access the contents of DOS/Basic data and text files. With the programs discussed here which allow access to both the directories and the files themselves, the entire package should send the users well on their way to complete conversion to the UCSD system.

What does all of that mean? It means that in North Star, your real variables can have a mantissa plus or minus 9.9999999 and an exponent ten to the plus or minus 64. UCSD has a mantissa plus or minus two to the 24 minus 1, and an exponent two to the plus or minus 128. This works out to an overall range of 9.9999999+ 64 to -9.9999999E-64 for North Star. UCSD has a range of about 0.1701411247E+37 to -0.1701411247E-37, but only about six to seven digits of the mantissa are significant. Note that the UCSD mantissa can exceed 0.17014—if the exponent is of less magnitude. The magnitude of both the mantissa and the exponent of North Star exceeds that of UCSD. This means that both a loss of significance and a possibility of overflow can occur in North Star to UCSD floating point data conversions.

Exponents on the order of ten to the 32 to ten to the 64 may not concern the user unless his data is of a highly scientific nature. However, the loss of significant digits in the mantissa is of concern in circumstances as every day as a seven digit phone number, or dollar amounts over $10,000.00. The user must deal with these problems on a case-by-case basis.

Real data is converted in CVTDATA by a function called CVTREAL. CVTREAL reads North Star real data through our old friend GETCHAR. It returns a variable real parameter containing the value. A boolean overflow flag is the result of the function. Upon entry to CVTREAL the variable VAL is initialized to zero. Four bytes are read via GETCHAR. Each byte is split into two BCD digits giving eight BCD digits total. VAL is multiplied by ten and a digit is added to it for all eight digits. The last digit or two may not add any significance to the mantissa due to the limitation discussed earlier. Because of the way the mantissa is built up, it will be necessary to divide by 1.0E08 later in the processing. The fifth byte is read and the mantissa sign bit is striped. The excess 64 is subtracted from the exponent, and its sign is determined. Overflow conditions are tested, and if they occur the value of VAL is set to a maximum. The exponent is either multiplied or divided into VAL and VAL is normalized. As usual, GETCHAR is called again to make ready for the next routine.

The main program begins by prompting the user for the input file name. Recall that the program is only a model which outputs to the CONSOLE: device, so no output name is requested. LASTCHR and BFPTR are initialized. GETCHAR obtains the first character.

While there are still characters to process, the following actions occur. If CHINT is greater than 15 real data is to be processed. A message that a number is being processed is printed, and an overflow indication (if necessary) follows that. Finally, the value obtained by CVTREAL is printed. If the original value of CHINT was less than or equal to 15, then a "case" statement is used to process the other options. Option one is an end of file. When North Star EOF is read, a message is printed and LASTCHR is set to true. This causes an exit from the "while" loop and the program terminates. Case two is a long string. Long strings have a two byte, low order byte first, length field. GETCHAR gets the first byte and it is stored in COUNT. GETCHAR gets the second byte, which is multiplied by 256 and added to COUNT. The third case is a short string. The length filed is a one byte field which is read by GETCHAR and saved in COUNT. After leaving the "case" statement, the string is output by a loop whose index runs from 1 to COUNT. The string is output to the CONSOLE: one byte at a time. CHINT values of zero or ten (i.e. nulls and line feeds) are skipped. This is not a necessary feature and can be eliminated if desired. The "one character at a time" output can be replaced by code which puts the characters in a packed array or string for further processing if needed.

The principals demonstrated in CVTDATA can be applied to any North Star data where the precision restrictions are not of concern. Listing 6 is a procedure called WRITREAL which can be incorporated into CVTDATA to replace CVTREAL with slight modification to CVTDATA. WRITREAL reads an eight digit North Star value and outputs it to the CONSOLE: in standard scientific notation. The user may be able to modify this routine to suit the need to extract all of the significance of real data. How he manages to further process this data in Pascal is up to him.

Summary

Some of the many advanced features now available to North Star Pascal users have been demonstrated in the directory conversion programs. The user of North Star/ UCSD Pascal now has all of the necessary tools to access the contents of his DOS/Basic data and text files. With the programs discussed above which allow access to both the directories and the files themselves, the entire package should send the users well on their way to complete conversion to the UCSD system. So go to it!

Listing 4

```
            (* Program to produce a U.C.S.D. text *)
            (*  file from a North Star BASIC text.*)
            (*  Written Aug. '80                  *)
            (*       by Chris Young               *)
            (*          3119 Cossell Drive        *)
            (*          Indianapolis IN 46224     *)
            (*             (317)-291-5376         *)
program CVTBAS;
 var KEY        (* used to map keywords into strings *)
       :packed array[0..255]of string[10];
     CH:char;
     BUFR:packed array[0..511] of char;
     I, J,      (* indicies *)
     CHINT,     (* contains ORD(CH) *)
     BFPTR      (* pointer into input buffer *)
       :integer;
     LASTCHR,   (* is true after last character is processed *)
     LABL,      (* if true, next 2 bytes contain a line number *)
     ISFILE     (* output flag *)
       :boolean;
     OUTNAME,   (* name of output device or file, if null then *)
                (*  output to console only                      *)
     INPNAME    (* name of BASIC file to be read *)
       :string[20];
     BASIC      (* file of BASIC text to be converted to *)
                (*   U. C. S. D. text                    *)
       :file;
     OUTDEV     (* file for converted text *)
       :text;
 procedure INIT1;
  begin
   (* initialize array of keyword tokens to default to   *)
   (* their ASCII values                                 *)
   for I:=0 to 255 do
    begin
     KEY[I]:=' ';       (* make strings of length 1 *)
     KEY[I][1]:=CHR(I); (* put in the ASCII value   *)
    end;
   (* now fill in keywords which are not one-to-one with ASCII *)
   KEY[128]:='LET';     KEY[129]:='FOR';
   KEY[130]:='PRINT';   KEY[131]:='NEXT';
   KEY[132]:='IF';      KEY[133]:='READ';
   KEY[134]:='INPUT';   KEY[135]:='DATA';
   KEY[136]:='GOTO';    KEY[137]:='GOSUB';
   KEY[138]:='RETURN';  KEY[139]:='DIM';
   KEY[140]:='STOP';    KEY[141]:='END';
   KEY[142]:='RESTORE'; KEY[143]:='REM';
   KEY[144]:='FN';      KEY[145]:='DEF';
   KEY[146]:='!';       KEY[147]:='ON';
   KEY[148]:='OUT';     KEY[149]:='FILE';
   KEY[150]:='EXIT';    KEY[151]:='OPEN';
   KEY[152]:='CLOSE';   KEY[153]:='WRITE';
   (* code 154 means a line number label follows *)
   KEY[154]:='';        KEY[155]:='CHAIN';
   KEY[156]:='LINE';    KEY[157]:='DESTROY';
   KEY[158]:='CREATE';  KEY[159]:='ERRSET';
   KEY[160]:='RUN';
 end; (* INIT2 *)
 {* the INIT procedure is too large so split it in two *)
 procedure INIT2;
  begin
   KEY[161]:='LIST';    KEY[162]:='MEMSET';
   KEY[163]:='SCR';     KEY[164]:='AUTO';
   KEY[169]:='NSAVE';   KEY[170]:='SAVE';
   KEY[171]:='BYE';     KEY[172]:='EDIT';
   KEY[173]:='DEL';     KEY[174]:='PSIZE';
   KEY[175]:='CAT';     KEY[176]:='STEP';
   KEY[177]:='TO';      KEY[178]:='THEN';
   KEY[179]:='TAB';     KEY[180]:='ELSE';
   KEY[181]:='CHR$';    KEY[182]:='ASC';
   KEY[183]:='VAL';     KEY[184]:='STR$';
   KEY[185]:='NOFNDMARK';       KEY[186]:='INCHAR$';
   KEY[187]:='FILE';    KEY[224]:='(';
   KEY[255]:=';'        KEY[226]:='*';
   KEY[227]:='+';       KEY[228]:='[';
   KEY[229]:='-';       KEY[231]:='/';
   KEY[236]:='AND';     KEY[237]:='OR';
   KEY[239]:='>=';      KEY[240]:='<=';
   KEY[241]:='<>';      KEY[244]:='<';
   KEY[245]:='=';       KEY[246]:='>';
   KEY[247]:='NOT';     KEY[198]:='INT';
   KEY[204]:='LEN';     KEY[205]:='CALL';
   KEY[206]:='RND';     KEY[202]:='SGN';
   KEY[203]:='SIN';     KEY[210]:='ATN';
   KEY[216]:='FREE';    KEY[217]:='INP';
   KEY[218]:='EXAM';    KEY[219]:='ABS';
   KEY[220]:='COS';     KEY[221]:='LOG';
   KEY[222]:='EXP';     KEY[223]:='TYP';
 end; (* INIT2 *)
 procedure GETCHAR;
  (* This procedure gets a character from the file BASIC and    *)
  (* returns the character in the global variables CH and CHINT.*)
  var N:integer;
  begin(*GETCHAR*)
   if not LASTCHR       (* if last character has *)
                        (* been read, don't try again *)
    then begin
          if BFPTR=512  (* buffer is empty, get new buffer *)
           then if EOF(BASIC)
                 then begin     (* no more buffers in file *)
                       BFPTR:=0;
                       BUFR[0]:=CHR(0);
                       LASTCHR:=true; (* this shuts off GETCHAR *)
                                      (* even if N* EOF has not *)
                                      (* been read              *)
                      end (* if EOF(BASIC) *)
                 else begin
                       N:=BLOCKREAD(BASIC,BUFR,1);
                       BFPTR:=0;
                      end; (* if not EOF(BASIC) *)
          CH:=BUFR[BFPTR]; CHINT:=ORD(CH);
          BFPTR:=BFPTR+1;
         end; (* if not LASTCHR *)
  end; (* GETCHAR *)
 procedure DOLABFL;
  (* This procedure is called after every occurence of the       *)
  (*  label flag which is code 154, and at the begining of every *)
  (*  line. It reads 2 bytes and outputs the integer value as a  *)
  (*  string of ASCII characters.  The value is interpreted as a *)
  (*  16 bit unsigned integer from 0 to 65535.  Note this is     *)
  (*  outside the range of Pascal integer variables, so real     *)
  (*  variables are used.                                        *)
  var LINENUM,  (* the value of the label as a real number      *)
      P         (* real power of ten  *)
        :REAL;
      LEADZERO  (* supresses output of leading zeroes    *)
        :boolean;
```

```
KEY[...]:='S...';    ...164]...TO'...
KEY[165]:='LOAD';    KEY[166]:='CONT';
KEY[167]:='APPEND';  KEY[168]:='REN';
```

```
begin
LEADZERO:=true;
```

```
  LINENUM:=CHINT;        (* low order byte  *)
  GETCHAR;               (* high order byte *)
  LINENUM:=LINENUM + 256.0 * CHINT;
  LABL:=false;           (* am no longer reading label *)
  for I:=4 downto 0 do
   begin
    P:=PWROFTEN(I);
    J:=TRUNC(LINENUM/P);          (* divide out high order digit *)
    LINENUM:=LINENUM - J * P;
    LEADZERO:=LEADZERO and (J=0);    (* LEADZREO true until J<>0 *)
    if not LEADZERO then
     begin
      if ISFILE then WRITE(OUTDEV,J);
      WRITE(J);
     end; (* if not LEADZERO *)
   end; (* for I *)
 end; (* DOLABEL *)
begin (* MAIN *)
 INIT1;  INIT2;
 WRITE('Type BASIC File-name:'); READLN(INPNAME);
 RESET(BASIC,INPNAME);
 WRITE('Type new file name:');   READLN(OUTNAME);
if LENGTH(OUTNAME)>0 then begin
                           REWRITE(OUTDEV,OUTNAME);
                           ISFILE:=true
                          end
                     else ISFILE:=false;
 LABL:=true;   (* file starts with a label *)
 LASTCHR:=false;
 BFPTR:=512;   (* this means buffer is empty *)
 GETCHAR;      (* skip length byte *)
 LASTCHR:=LASTCHR or (CHINT=1);       (* test for N* EOF *)
 while not LASTCHR do
  begin
   GETCHAR;
   if LABL
    then DOLABEL
    else begin
          LABL:=(CHINT=154) or (CHINT=13);
          if ISFILE then WRITE(OUTDEV,KEY[CHINT]);
          WRITE(KEY[CHINT]);
          if CHINT=13 then
           begin
            GETCHAR;          (* skip length byte *)
            LASTCHR:=LASTCHR or (CHINT=1);    (* test for N* EOF *)
           end;(*if CHINT=13*)
         end;(*if LABL*)
  end;(*while*)
 CLOSE(OUTDEV,LOCK);
end.
```

Listing 5

```
              (* Program to demonstrate North Star  *)
              (*  to U.C.S.D. Pascal data conv.     *)
              (*  Written Aug. '80                  *)
              (*       by Chris Young               *)
              (*          3119 Cossell Drive        *)
              (*          Indianapolis IN 46224     *)
              (*             (317)-291-5376         *)
program CVTDATA;
 var CH           (* character returned by GETCHAR *)
           :char;
     BUFR       (* input buffer used by GETCHAR   *)
           :packed array[0..511] of char;
     I, J,      (* indicies *)
     CHINT,     (* contains ORD(CH) returned by GETCHAR *)
     COUNT,     (* length of string *)
     BFPTR      (* pointer into input buffer *)
           :integer;
     LASTCHR    (* is true after last character is processed *)
           :boolean;
     VALU       (* temp real *)
           :real;
     DOS        (* N* DOS format data file for input *)
           :file;
     INPNAME    (* name of DOS file to be read *)
           :string[20];
procedure GETCHAR;
 (* This procedure gets a character from the file DOS and *)
 (*   returns the character in the global variables *)
 (*   CH and CHINT.             *)
 var N:integer;
 begin(* GETCHAR *)
  if not LASTCHR        (* if last character has *)
                        (* been read, don't try again *)
   then begin
         if BFPTR=512  (* buffer is empty, get new buffer *)
          then if EOF(DOS)
                then begin     (* no more buffers in file *)
                      BFPTR:=0;
                      BUFR[0]:=CHR(0);
                      LASTCHR:=true; (* this shuts off GETCHAR *)
                                     (* even if N* EOF has not *)
                                     (* been read              *)
                     end (* if EOF(DOS) *)
                else begin
                      N:=BLOCKREAD(DOS,BUFR,1);
                      BFPTR:=0;
                     end; (* if not EOF(DOS) *)
         CH:=BUFR[BFPTR]; CHINT:=ORD(CH);
         BFPTR:=BFPTR+1;
        end; (* if not LASTCHR *)
 end; (* GETCHAR *)
function CVTREAL(var VAL:real): boolean;
  var EXPSGN,   (* sign of exponent, +1 or -1    *)
      HI,       (* high order 4 bit BCD value of a byte *)
      LO,       (*  low order 4 bit BCD value of a byte *)
      I,        (* index *)
      EXPON     (* exponent as integer *)
         :integer;
      A         (* real value of ten to the EXPON *)
         :real;
    OVFL:boolean;
begin
 VAL:=0;
 for I:=0 to 3 do      (* read 4 bytes, 8 BCD digits *)
  begin
   HI:=CHINT div 16; (* high order 4 bits *)
   LO:=CHINT mod 16; (*  low order 4 bits *)
   VAL:=VAL * 100 + HI * 10 + LO;     (* build up value *)
   GETCHAR;
  end;
 if CHINT>127 then     (* high order bit of 5th byte *)
                       (*  is sign of mantissa       *)
```

```
  begin
   CHINT:=CHINT-128;
   VAL:=-VAL;
  end;
 EXPON:=CHINT;
 if EXPON<>0          (* zero exponent means zero value *)
  then
   begin
    EXPON:=EXPON-64; (* exponent is excess 64 *)
    if EXPON>0 then EXPSGN:=1 else EXPSGN:=-1;
    EXPON:=ABS(EXPON);
    (* Note:  0.1701411247E+-37 is the limit before overflow *)
    OVFL:=(EXPON>37)or((EXPON=37)and(VAL>17014112.0));
    if OVFL then begin EXPON:=37; VAL:=17014112.0 end;
    A:=PWROFTEN(EXPON);
    if EXPSGN<0 then A:=1.0/A;
    (* VAL is on the order of 1E8 so normalize it and *)
    (*  multiply in exponent.                         *)
    VAL:=VAL/1.0E8 * A;
   end (* if EXPON<>0 *)
  else
   begin
    VAL:=0; OVFL:=false;
   end;(*if EXPON=0*)
 GETCHAR;     (* always leave a character in CH and CHINT *)
 CVTREAL:=OVFL;
end;(*CVTREAL*)
begin (* CVTDATA *)
 WRITE('Type DOS data file name:'); READLN(INPNAME);
 RESET(DOS,INPNAME);
 LASTCHR:=false;
 BFPTR:=512;    (* this means buffer is empty *)
 GETCHAR;       (* skip length byte *)
 LASTCHR:=LASTCHR or (CHINT=1);       (* test for N* EOF *)
 while not LASTCHR do
  begin
   if CHINT>15 (* this means it is a real value *)
    then begin
          WRITE('Number =');
          if CVTREAL(VALU) then WRITE('**** OVERFLOW **** ');
          WRITELN(VALU);
         end (* if CHINT>15 *)
  else
     begin
      case CHINT of
       1: begin
           WRITELN('EOF');
           LASTCHR:=true;
          end;
       2: begin
           WRITE('Long string ');
           GETCHAR;
           COUNT:=CHINT;
           GETCHAR;
           COUNT:= 256 * COUNT + CHINT;
           GETCHAR;     (* always leave a character *)
                        (*   in CH and CHINT        *)
          end;
       3:begin
          WRITE('Short string ');
          GETCHAR;
          COUNT:=CHINT;
           GETCHAR;     (* always leave a character *)
                        (*   in CH and CHINT        *)
          end;
       end; (* case *)
      if not LASTCHR then
       begin
        WRITE('"');
        for I:=1 to COUNT do
         begin
          if (CHINT<>0) and (CHINT<>10) then WRITE(CH);
          GETCHAR;
         end; (*for I:=1 to CHINT *)
        WRITELN('"');
       end; (* if not LASTCHR *)
     end; (* if CHINT>15 then ... else *)
  end; (* while not LASTCHR *)
end. (* CVTDATA *)
```

Listing 6

```
            (* Program to output North Star 8 dig.*)
            (*  real to CONSOLE:                  *)
            (*  Written Aug. '80                  *)
            (*       by Chris Young               *)
            (*          3119 Cossell Drive        *)
            (*          Indianapolis IN 46224     *)
            (*             (317)-291-5376         *)
procedure WRITREAL;
(* Reads North Star 8 digit real and outputs it to CONSOLE:   *)
 var I,       (* index *)
     EXPON    (* exponent as integer *)
         :integer;
     MANTSGN  (* if true, mantissa is negative *)
         :boolean;
     DIGITS   (* array of 8 digits *)
         :packed array[0..7] of char;
 begin
  for I:=0 to 3 do    (* read 4 bytes, 8 BCD digits *)
   begin
    DIGITS[I*2]:=CHR((CHINT div 16)+48);   (* high order 4 bits *)
    DIGITS[I*2+1]:=CHR((CHINT mod 16)+48); (*  low order 4 bits *)
    GETCHAR;
   end;
  if CHINT>127  (* high order bit of 5th byte is sign of mantissa *)
   then
    begin
     CHINT:=CHINT-128;
     MANTSGN:=true;
    end
   else MANTSGN:=false;
  EXPON:=CHINT;
  if EXPON<>0          (* zero exponent means zero value *)
  then
   begin
    EXPON:=EXPON-64; (* exponent is excess 64 *)
    if MANTSGN then WRITE('-') else WRITE('+');
    WRITE('0.');       (* output first digit and decimal pt. *)
    for I:=0 to 7 do WRITE(DIGITS[I]);  (* output digits *)
    WRITE('E',EXPON);
   end (* if EXPON<>0 *)
  else
   WRITE('0.000000E+00');
 GETCHAR;     (* always leave a character in CH and CHINT *)
end;(*WRITREAL*)
```

Chapter IV
Software Reviews

The Mate Text Editor — Word Processor

R.D. Graham

One of the widely known benefits accruing to the S-100 computer owner is that he can use the CP/M™ operating system. One of the widely unknown benefits of using CP/M is the opportunity to use a suberb text editor-word processor called MATE. This program, designed and coded by Michael Aronson, was copyrighted early in 1979 and sold, until recently, for $69.50. Aronson, for some reason, did not aggressively market his product and it has become known to only a small sample of the CP/M users through word-of-mouth.

I have used it for six months now, writing all my assembly, FORTRAN, CBASIC, PILOT and C source programs with it and would feel crippled without it. Two years ago I had a hard time getting used to the CP/M editor (ED) but finally grew rather fond of it. However, since acquiring MATE, I have not written a program with ED. One thing more, which I'm sure will surprise Pencil enthusiasts, (and I include myself amont them); I rarely use Pencil now, finding MATE my choice in word processing, although MATE does not have all the output formatting capabilities present in PENCIL.

MATE comes with a good user manual and interface guide, and Aronson shows that he is sensitive to the documentation problem by the way he has designed and written it. The disk I received had drivers for interfacing with VDM-1, ADM-3 and Hazeltine 1500 CRT's in both HEX and ASM files. Following clear instructions in the interface guide of the manual, I had my VDM-1 version up and running without trouble.

Aronson, in the introduction to his manual, spells out the commonly accepted meanings of "text editor", "word processor" and "text output processor" and explains that "MATE is an attempt to combine some of the best features of all three". I think he has been successful in this attempt.

R.D. Graham, 550 Starboard Drive, Naples, FL 33940

Mate is available from: Michael Aronson, Aox Inc., 14 East St., Hopkinton, MA 01748; (617) 435-4840.

Mate comes up in a "Command Mode" which is reminiscent of CP/M's ED. There are a wealth of commands here, the majority of which, I must confess, I don't use much because I find it so convenient to use similar "instantaneous" commands in the "Insert Mode". In this mode, what you see is what you get. Text is entered by simply typing. Editing changes show up instantaneously on the screen at the cursor position. No more blind editing! You can move the cursor to the beginning or end of the text buffer with control A and control Z respectively, and besides moving the cursor up or down one line at a time you can move it up or down 6 lines at a time. This allows you to move through your text very rapidly. Similarly you can move the cursor forward or backward one character at a time, or one word at a time. Insertion or deletion of text at the cursor is similarly easy, instantaneous and always with the sure knowledge that it has been done correctly since you see it happen. Big blocks can be moved either with tags or with easy moves of text to one of ten text buffers available, from which it is inserted at the cursor position with another simple command.

Search, search and change, set tab stops, delete tab stops, set left and right margins are all commands (with many options) available to the user of MATE.

Users with big complicated editing jobs will probably find the macro facilities available in MATE very much to their liking for they can, in effect, add their own commands to MATE's command set. To aid in "programming" these complex macro command strings, Mate includes a breakpoint and trace facility. I have not attempted to build any macro command strings because for the uses I make of MATE I find it quite powerful enough the way it is. However, many will probably want to improve its output formatting capability and this would be one way to do so.

In summary, I think that for the money MATE cannot be beat; and that many of you will agree with me that in preparing source code files under CP/M it cannot be beat at any price.

Information Master

Bill Machrone

Information Master is a CP/M*—compatible information retrieval program oriented towards textual data. The program, available from Island Cybernetics, was originally written to perform retrieval from a large data base of articles and abstracts in the ecological sciences field. It, however, is a generalized program and is adaptable to a number of different retrieval needs.

Information Master operates under a variety of the CP/M-derivative operating systems, such as CDOS and IMDOS. It is "installable," in that some of the operating parameters can be changed for specific applications. The program is fast, since it maintains the dictionary in main memory during retrieval. The console displays are not sophisticated in that there are no cursor controls or even screen clears, but capabilities of that type are usually just window dressing, anyway. It does, however, format text going to the printer and gives you the option of sending it to a disk file for further editing or processing. Within its defined area of operation it does quite a lot, especially for the price, which is $37.50 per copy.

One of the unique things about Information Master is that it does not have the input/data entry module that is usual for this type of program. Since its major function is to facilitate access to text, it is specifically intended for use with your current text editor. One advantage to this approach is that it isn't necessary to learn a new set of text editing rules in order to use Information Master. Whatever CP/M-compatible text editor you are familiar with is fine. There is also no reason why you can't use whatever high-level language you have at hand to create prompted input acceptable to Information Master. The files of information you create are considered "raw" text by Information Master; it processes them to build a dictionary of retrieval terms and a pointer file that provides access to the text.

Island Cybernetics provides a demonstration data base with the programs, and it is worthwhile to experiment with it before you plunge into creating your own. The data base has extracts from articles which are cross referenced by the topics upon which they are likely to be retrieved. A feature of the program is that only the dictionary and pointer files need be present on one disk. The data itself may be on a completely separate disk, thus maximizing data storage.

The input requirements are simple. There are three "triggers" or delimiters that Information Master looks for in raw text in order to distinguish keywords from text. One of the dilimiters is used to establish a "brief" retrieval heading, such as the title of an article. Below is an example from the Information Master manual:

```
*C
INFORMATION MASTER, Users Manual, Island Cybernetics, 1979 *This short manual describes the
use of the "INFORMATION MASTER" program
for retrieval of text files using Boolean combinations of key words or phrases.
Vendor:
  Island Cybernetics
  P.O. Box 208
  Port Aransas, TX 78373
*K
INFORMATION RETRIEVAL/CP/M/DATA
MANAGEMENT/8080 CPU Z-80 CPU
*E
```

If the above entry (and any number of similarly organized entries) is presented to Information Master as raw text, it will be cataloged and cross-indexed by the keywords that follow the *K delimiter. The *E signifies the end of the entry. If you use the "short form" of retrieval, the program will display the text from the *C to the first*. If you specify the long form, it will display all of the text down to the *K.

Another nice feature of Information Master is that the output can be directed to either the list device or the console. While we're on the topic of nice features, another that deserves mention is the "not in dictionary" function. If you request a lookup under "Z-80" the program will inform you that there is no corresponding entry in the master dictionary and will then list the close matches to the entry you had specified. This makes it easy to pick out the entries you want. The "sounds like" algorithm may be a little generous in terms of giving you

Bill Machrone, P.O. Box 291, Fanwood, NJ 07023

*CP/M is a registered trademark of Digital Research

some matches that aren't even close to what you want, but it's better to have too many than too few.

Actual retrieval from the data base is done by specifying the keywords that you are looking for. Information Master provides a Boolean expression input capability, so that you can logically AND and OR your requirements. This feature alone sets it apart from the usual data retrieval applications written in Basic, which normally do not provide this function. Furthermore, most homegrown retrieval systems are limited in the number of keys that can be stored or retrieved upon. There is no limit to the number of keywords that can be associated with each piece of information, so that the cross-indexing capabilities are endless.

Now that you know what Information Master does, the inevitable question arises, "What good is it?". Most of us don't have large data bases of articles and books to summarize, but we do have some commonplace data that could stand some organization, and there is the occasional unique application that can benefit from a program such as this. The manual contains some suggestions in addition to data bases of literature, including book collections, correspondence and recipes.

Taking recipes as an example, you can enter your favorite dishes and document where the recipes are located and what variants you have tried. Below is an example of how you might organize these entries:

```
*C
Chicken with walnuts in plum sauce
Bon Appetit, July, 1980 Page 8.
*Use 30% more sauce than recipe calls for. Breast
meat a good substitute for thighs. Goes well over
fried rice and with pina coladas. Simple but impres-
sively good.
*K
CHINESE/CHICKEN/GINGER/HOISAN/WAL-
NUTS/DINNER
*E
*C
Oven fried fish
Better Homes Cookbook, page 260.
*Season bread crumbs with parsley, bouquet
garni, parmesan cheese, dash garlic salt, tarragon,
basil, oregano, or whatever comes to mind. 8-10 min-
utes sufficient for thin fillets.
*K
FISH/DINNER/FAST
*E
```

Information Master's short and long form output enables you to list just the recipe titles and the publication or list your comments as well. Any number of entries such as the ones above can be present in the raw text file. Information Master provides the dual advantage of random access with variable length records for the most efficient possible utilization of your disk storage. If you carefully standardize the usage of keywords you will have no trouble retrieving whatever you want from the data base. For example, you can specify "FAST and DINNER," "CHINESE and CHICKEN and DINNER," or something like "CHICKEN or FISH and DINNER."

A totally different potential application is a personal diary or a businessman's calendar. In this mode, you could use the keywords to establish the date, the type of event and meaningful cross-indexes. The short form entry need not be used. Here's an example:

```
*C
10:00 Meeting with Joe Tyler. Discussed new appli-
cations program and suggested that Steve Linden
be appointed as user liaison. Tyler not sure about
Linden; will get back to me by 30 June.
*K
MEETINGS/10/JUN/1980/APPLICATIONS/
TYLER
*E
*C
1:30 Phone with C. Daniels of Hairy Software Inc.
Determined availability of King Kong word
processing system. Version 1.0 will be replaced
in 45—60 days. Field upgrade to existing
licensees is for cost of media and manuals.
*K
10/JUN/1980/KING KONG/WORD PROCES-
SING
*E
```

In this kind of example, Information Master can manage past or future appointments and to-be-done items. With a little ingenuity, follow-up dates could be coded as part of the keyword area, so that an inquiry can tell you almost instantly what needs to be done by a certain date. Anyone can appreciate the permanence of the records and the ability to review a month's meetings or all those held on a given topic or with a specific individual.

In conclusion, Information Master is unique in its "cataloging" capabilities of text and is adaptable to a variety of storage and retrieval needs. If you don't need to do a lot of field-oriented further processing with the retrieved data and if simple list or console output is sufficient, Information Master can do things that would other wise take extensive custom programming or cost far more for a generalized data base management subsystem. I think that it compares very favorably to data managers like WHATSIT and Selector III, especially considering the price. This is not to say that it would replace either of them; WHATSIT is uniquely capable in expressing hierarchical relationships among data items, while SELECTOR has a full range of report generation capabilities that are quite powerful in themselves. I feel, however, that neither of them could beat Information Master at its own game. It doesn't resort to cute "artificial intelligence" conversations with the user and, depending on how you set up your keywords, can represent hierarchical or relational data structures. It would be nice to see some substring operators so that it wouldn't be necessary to break up the year, month and day, and so you could pick out subcodings like "CPU" from both "Z-80 CPU" and "8080 CPU." A negation operator would be neat, too. Then you could say, in essence, "DINNER but not FISH."

But all this is quibbling. Information Master is a good buy, has no apparent bugs, is reasonably well documented and is both easy and fun to use. It is available from: Island Cybernetics, P.O. Box 208, Port Aransas, Texas 78373, tel: (512) 749-6673. The cost is $37.50.

MODKOM

Dennis Thovson

MODKOM is a set of programs that allows you to use your computer to communicate with other computers via a modem and the telephone system. Included in the package are programs that permit your computer to function as a local terminal to a remote computer system (or in a conversational mode with another computer) and a complimentary set of programs to transfer files between computers. The programs are available from Data Systems Inc. and are written in 8080 assembly language for the CP/M operating system.

These programs provide a general purpose CP/M communications capability. All input/output (I/O) from the programs are handled by calls to the console, reader and punch through the standard CP/M I/O BDOS entry at 05H. This technique allows these programs to run on any system using CP/M. However, the penalty paid for universal applicability, as implemented in MODKOM, is that you are restricted to a 7-bit plus parity ASCII format for all data transmitted. This is not too severe a penalty, for a couple of reasons. First, most information transferred by the average microcomputer user is text or ASM files, and second, there is a technique for transmitting 8-bit information (e.g., COM files) via the standard hex format which converts each 8-bit byte into two ASCII characters. The standard hex format can be converted back to 8-bit information by using the CP/M LOAD command.

The terminal program is called "Converse" and its principal function is to permit terminal-to-terminal communication between two computers. There are a number of modes which can be toggled on and off by entering specific control characters while in the Converse program (the control characters are not sent to the distant terminal). A set of control characters enables you to: transmit a named CP/M file to the distant computer system; save all incoming information in memory and subsequently write the saved information to a named CP/M file; and send incoming information to the printer. Control D toggles a full/half-duplex software switch. In the half-duplex mode, all characters entered at your terminal are echoed to your console. In the full-duplex mode, characters are not echoed locally, so if you want to see the characters entered at your terminal, the remote system must echo its received characters to you. (Most time sharing systems echo all received characters.) A control G (Goodbye) returns you to CP/M.

Files transmitted while in the Converse program are sent without any error checking and as a single block. Therefore, the receiving end must be able to accommodate the entire incoming file in memory if no information is to be lost. Large files transmitted in this manner will have to be broken into smaller files and transmitted individually. Transmission of large files is better handled by the specialized file transfer programs called "Transmit" and "Receive."

Transmit and Receive are complementary programs for transferring CP/M files in a block mode with error checking. The command line syntax for both TRANSMIT and RECEIVE includes, respectively, the source and destination file names. The Receive program must be ready and waiting before the Transmit program starts to send the data. Files are sent in 2K blocks with no handshake protocol between the transmit and receive ends. The sequence of operations is as follows: the Transmit program reads a 2K block from the file, sends an ASCII "STX" (start of text) and starts transmitting the 2K block of characters. At the receive end, the Receive program detects the STX and reads the incoming 2K block of information into memory. When the Receive program has received 2K characters, it writes them to disk, resets its pointers and looks for the start of another block. While the Receive program is writing that 2K block to disk, the Transmit program reads in the next 2K block and transmits a predefined number of nulls. The Receive program ignores the nulls while waiting for the start of the next block. This sequence continues until the end of the file is reached. At the end of the file, the Transmit program sends an ASCII "ETX" (end of text) and a checksum for the entire file. The Receive program detects the ETX, calculates its own checksum, compares it with the received checksum and informs the user if the transfer was successful or not.

Dennis Thovson, 243 McMane Avenue, Berkeley Hts., NJ 07922.

This type of file transfer sequence is an open loop system, i.e., the transmit and receive ends run independent of one another. This imposes some constraints on the timing of the data transfer to ensure that the receiver is ready when the transmitter starts to send data. In this case, the number of nulls sent by the transmit end has be be chosen to allow sufficient time for the receiving end to write the previous 2K block to disk before the next block is transmitted. If the disk systems at both ends are similar, the time difference between reading and writing a 2K block ought to be small, so the number of nulls required should be small. However, to be safe with an unknown system, the instructions recommend setting the number of nulls to 200.

A pair of programs called "Unload" and "Hexcheck" are included as utility programs to handle non-ASCII files such as COM files. Unload converts 8 bit information into the standard hex format and saves it as a disk file. This file can then be transferred by the Transmit and Receive programs. The Hexcheck program reads a disk file containing a HEX file and calculates the checksum for each line of information. This intended as an error check on a received hex file. A hex file can be converted back to 8 bit information using the standard CP/M LOAD utility.

All programs are furnished in 8080 assembly language and can be assembled using either the standard CP/M ASM or MAC. The program developers intend that you modify your CP/M BIOS to incorporate the console I/O, reader and punch routines to handle the CP/M IOBYTE logical to physical device assignment. Also, your modem status, input and output routines will have to be integrated into your BIOS. The documentation furnished with MODKOM contains two sample BIOS listings as a guide to the modifications required. If you do not want to modify your BIOS, a program called "Mkbios" is furnished which must be assembled as part of each of the main programs, Converse, Transmit and Receive. At run time Mkbios replaces your BIOS jump table console related entries with jumps into Mkbios instead of your normal console routines. Your modem I/O requirements will have to be incorporated into MKbios for this method of operation.

If you modify your BIOS as instructed, there is a mode available which allows you to put your computer into a condition where it can be controlled directly by a remote terminal operating in the Converse mode. The remote terminal then functions exactly the same as your own local console. This mode is not available with the Mkbios option.

You must know and be comfortable with assembly language and your CP/M BIOS (or know someone who is) to bring these programs up. The documentation received was an early version and, while complete, was quite difficult to decipher. In talking with Fred Lepow, who wrote the programs, we discussed a different method of presentation for future documentation which should make it easier to understand.

I have used the primary MODKOM programs and they all function as intended by the authors. My principal use has been as an intelligent terminal on a time-share system. Files have been successfully transferred both up and down without any difficulty. Some of the console handling routines required modification for user convenience and these were passed along to the authors for consideration in future releases of MODKOM. Source code is furnished for all programs so you can, of course, add your own favorite features. One feature that I would like to see as standard is Transmit and Receive callable options from Converse so that large files can be transmitted without having to worry about available memory at the receive end. Also, in the normal Converse file transfer mode, I would like to be able to interrupt the file transfer at any point rather than lose control until the entire file is transmitted.

These programs fill a definite need in the expanding world of communications between micro computer users. The approach taken by the authors was to stay strictly within CP/M conventions so Converse, Transmit and Receive would be portable, without modification, to any CP/M system. These are no-frills programs that provide a basic communications capability that should meet most user needs. However, there are two areas that I think could be improved—file transfer synchronization and error checking. Although the open loop method of transferring files works with relative safety and without much loss of efficiency, it could be improved by implementing a simple "hand-shake" protocol for each block of data transferred. For example, Ward Christensen's MODEM program in the CP/M User's Group Library sends one sector along with a checksum and waits for an acknowledge (ACK) from the receiving end before transmitting the next sector. This solves two problems: The receiving end can delay sending an acknowledge to the transmitter until it finishes writing a block to disk, thus eliminating any synchronization problem between disk operations at either end; and any transmission errors are quickly detected so the sector can be immediately retransmitted in file sequence. The MODKOM approach of sending a single checksum for the entire file instead of for each block probably works well most of the time at 300 baud. However, it seems to me that block error checking has very little, if any, disadvantage and it certainly has potentially significant advantage.

MODKOM is available in 8 inch single density and 5 inch North Star disk formats. It can be purchased from: Datastat Systems Inc., 631 B Street, San Diego, California 92101, tel: (714) 235-6602. The cost is $60.00

COMMX and MCALL — Two Terminal and File Transfer Programs

Glenn A. Hart

The growing interest in telecommunications with personal computers has led to a proliferation of software products designed to make the information interchange easier and more reliable. Programs are available for almost any hardware configuration and offer many features not previously available. These two programs are designed for use with a wide spectrum of systems running the CP/M and MP/M operating systems.

COMMX

COMMX acts as an intelligent interface between a personal computer and a time-sharing system, or between two personal computers. In addition to normal "dumb terminal" communications with a host system, COMMX handles file transfers both to and from the host system if the system supports the standard X-ON/X-OFF start/stop protocol. COMMX has provisions to route the entire dialog with the host system to a local disk file for later reference, allowing for "post-mortem" analysis of the session.

As with most such programs, communication between two personal computer systems requires that *both* systems operate with COMMX. Full conversational intercommunications are possible between the two systems, as are file transfers in either direction. A 16-bit CRC checking protocol is used as described later. COMMX will operate with either CP/M 1.4 or 2.2 (determining which version is being used and adjusting its operation accordingly) or with MP/M.

While the full source code for COMMX is available from the supplier, Hawkeye Grafix (23914 Mobile Street, Canoga Park, California 91307) for $250, most purchasers will undoubtedly order the object code only version which costs $75. This is an obvious attempt by Hawkeye Grafix to discourage purchase of the source, which does allow for the configuration to systems other than that of the purchaser, or at least to maximize their income from the dissemination of the full source code. I have not seen the source, but Will Pierce of Hawkeye indicates that it is heavily documented, with over 50K of code assembling down to the 6K executable object file. In any event, the buyer of the object code must specify the hardware environment in which COMMX will operate, including what I/O board will be used. Hawkeye will provide *two* implemented versions to a buyer of the object code version for the same $75 if both hardware configurations are specified when the program is ordered. COMMX versions are available for a broad spectrum of boards and systems listed in Table I. Documentation is direct and to the point (some might say a bit sparse) and clearly indicates how to use the program.

If the PMMI or Hayes versions are used, COMMX offers three initial options: answer, originate without dialing, or auto dialing. If auto dialing is chosen, the program dials a number input at the console and establishes the connection. An interesting feature is that the program works correctly with either 2 MHz or 4 MHz clock speeds automatically; it determines the system clock rate and adjusts internal timing loops.

Table I:
COMMX Available Configurations

Solid State Music 4SIO
IMSAI MIO and 2SIO
Micro-Da-Sys 4P4S
Delta Products CPU board
Apple Z-80 Slot 2 serial cards
Apple D. C. Hayes Micromodem
S-100 D. C. Hayes Micromodem
Vector Graphics Bitstreamer 2
Industrial Micro Systems 440
Datapro I/O Master
SuperBrain
TRS-80 Model II
PMMI modem card

Glenn A. Hart, 51 Church Road, Monsey, New York 10952.

Once the connection is made, the command mode menu shown in Table II is displayed. Most of the modes are self-explanatory. If the local mode is chosen, the menu in Table III is displayed. In Local Mode several choices duplicate standard CP/M functions without the need to leave COMMX. This is quite convenient, since these functions can be executed while the connection to the host computer is maintained. Turning console echo off can be useful if high speed file transfers are being made between two computers in the same room over a direct three-wire serial channel and a slow terminal is in use, or if a particular memory-mapped video display system would result in lost characters on the communications port while the CPU is scrolling the display. Such transfers can be made at speeds up to 9600 baud, and console echoing would obviously not allow such rates. The newest version of COMMX (6.0) displays a period for each 128 byte block transferred, which allows the user to know that things are progressing during such operations.

Table II
COMMX Command Mode Menu

COMMAND MODE FUNCTIONS:

1) LOCAL MODE
2) TERMINAL MODE
3) COPY HOST TO LOCAL
4) COPY LOCAL TO HOST
5) CONVERSATIONAL MODE
6) COPY COMM HOST TO LOCAL
7) COPY LOCAL TO COMM HOST
8) EXIT

Table III
COMMX Local Mode Menu

LOCAL MODE FUNCTIONS:
PRESS A THRU H FOR DIRECTORY OR:

1) COMMAND MODE
2) RENAME FILE
3) DELETE FILE
4) LOGIN NEW DISKS
5) CONSOLE ECHO IS: ON
6) TERMINAL MODE LOG
7) CTRL CHARACTER DISPLAY IS: OFF
8) 8 BIT DATA ENABLED

The control character display option is interesting. It causes all received control characters to display as the standard carat sign (^) plus the letter of the control character. This can be useful in unusual situations to determine what control characters are being transmitted.

COMMX allows direct transfer of full 8-bit files, but only if the serial board used supports more than the ASCII minimum 7 bits and the board has been correctly configured for such use. When set up this way, COM, INT and REL files can be transferred directly, without the need to use the supplied UNLOAD program to convert the file to HEX format.

Returning to Command Mode, normal terminal mode operates in full-duplex mode only, which is usually the desired method. As mentioned, terminal mode can be exited at any time, local mode operations performed and terminal mode resumed without losing the connection.

File transfers from the time-sharing host use the standard Control-S/Control-Q handshaking. A Control-G from the host is interpreted as an end-of-page indicator; the Control-G is not entered as data but is automatically acknowledged with a Control-Q to continue the transmission. This method worked on three time-sharing systems I tried; I gather these signals are quite standard on mainframe systems, but this is no guarantee of universality.

COMMX automatically allocates the largest disk buffer available given the user's memory. When the disk buffer fills up, COMMX sends a Control-S to stop the transmission, watching the line for a brief time to pick up any characters sent before the host stops transmitting. The buffer is written to a disk and a Control-Q sent to the host to resume sending. If the host does not recognize the standard handshaking, the size of the file which can be held in the disk buffer is about 50K bytes in a 64K system; this is the largest file downloadable without loss of characters in a non-handshaking environment. COMMX does not determine the end-of-file situation; the user must enter a Control-E to end the download.

Sending a file to the host works similarly. Whenever the commands that are necessary to set up the mainframe for accepting data are issued, the file to be sent is specified and transmission begins, continuing until the CP/M end-of-file marker Control-Z is encountered. COMMX prints a message to indicate that the complete file has been sent and the operator issues the necessary commands to the host system to close the newly input file.

Communication between two COMMX computers is even easier and more accurate. The conversational mode causes data keyed to be echoed to the terminal and simultaneously sent to the connected system. File transfers are similar to the host methods described above except that file operations are more automatic and a Cyclic Redundancy Check 16 protocol is used to assure perfect transmission and reception. The program calculates an ASCII equivalent of a full CRC 16 check as used in IBM 2780/3780 protocol and sends ACK or NAK signals to indicate good or bad transmission/reception. The program will re-try up to seven times to insure accuracy; after seven failures the operators are notified of the probability of a bad line. If an entire file is transmitted to its conclusion, it is a virtual certainty that the file has been conveyed with 100% accuracy.

MCALL

MCALL is available in two distinct versions, one for a wide spectrum of serial boards and computer systems, the other, designated AMCALL for its automatic dialing functions, specifically designed for either the PMMI or IDS modem boards. A version for Hayes Microcomputer Products boards is in development. The differences between the two versions are much more basic than the target hardware environment. While both function more or less similarly, the standard MCALL is written in 8080 assembly language while the modem board version is written in BDS C.

Both versions are provided in full source code. MCALL costs $85 and AMCALL $95 from Micro-Call Services.

9655-M Homestead Court, Laurel, Maryland 20810. I have never seen a better documented source listing than the MCALL assembly listing. It is virtually a tutorial on both good assembly language programming techniques and how UARTS and telecommunications hardware work. An indication of how extensive the comments are is the fact that the source the file is over 104K bytes and produces an executable COM file of only 9K.

The AMCALL version is a veritable "Rosetta Stone" for those not yet familiar with this excellent C language. Tim Pugh, MCALL's author, mentions that he was attempting to verify the assertion that C could replace assembly in systems type programming. AMCALL is a very positive indication that is indeed the case. Tim also includes an interesting discussion of the relative merits of C versus Pascal and other high level languages. He focuses on C's superior handling of pointers, and his arguments seem compelling.

I have never seen a better documented source listing than the MCALL assembly listing. It is virtually a tutorial on both good assembly language programming techniques and how UARTS and telecommunications hardware work.

The documentation provided with each version is excellent—detailed and informative. AMCALL is sold configured for either of the modem boards now supported and is ready to run as provided. Full instructions on modifying MCALL for the user's hardware configuration are provided; anyone with even the most rudimentary knowledge of assembly language should have no problem. All that is necessary is to set flags to indicate the system clock rate, default duplex mode desired (the default can be changed during operation), the number of retransmissions allowed during transfers and the CRT screen clear code.

In addition to these parameters, two other choices must be made. Both versions of MCALL offer a choice of two buffering methods for file transfers. The normal mode is called Big Buffer, and works exactly like the COMMX method; the entire TPA is available to hold incoming data - when this fills up, the host system is paused while the buffer is written out to disk. Big Buffer mode works perfectly with most host systems that accept standard handshaking, and the user could patch in non-standard handshaking characters into the MCALL source if necessary.

If the host system doesn't use *any* handshaking, the Double Buffer mode may work. This mode requires the host to send seven null characters at the end of each data line. MCALL maintains two buffers. While one is filling up, the other is written to disk during the sending of the nulls. This is dependent on critical timing, but can work. It is less desirable than the normal Big Buffer mode because of the constant disk accesses and somewhat touchy nature of the whole process.

MCALL also offers provisions for loading under DDT or SID. If a flag is set during assembly and the resulting MCALL file loaded with DDT or SID, a special escape character will transfer control from MCALL to the debugging program used.

Table IV
Boards/Systems supported by MCALL

The following systems/boards are currently supported:

1. TDL System Monitor Board (SMB)
2. Cromemco TU-ART board
3. INFO 2000 DISCO controller board
4. TEI Processor Terminal
5. JADE I/O Board
6. HEATH H-8
7. IMSAI MPUB processor board
8. Vector Graphics Bitstreamer II
9. SCION Microslice (Wordsmith computer)
10. AMD AmSYS 8/8 Microcomputer Dev. Sys.
11. SD Systems SBC-100/200
12. SSM I/O-4
13. Processor Technology (PTC) 3P+S
14. Digital Group System
15. Industrial Micro Systems IMS-440 I/O board
16. TRS-80 Model II with P&T CP/M 2.2
17. Godbout Interfacer II

MCALL supports a very large number of systems and/or boards, listed in Table IV. By virtue of supporting these boards, almost all UART's used in microcomputers are also supported (Table V). The documentation explains how to configure MCALL for a system not shown if the serial device is supported; if not, Tim Pugh will configure the unusual system for a modest fee.

When MCALL is invoked, a summary of the default configuration is displayed and the user is given the option of changing duplex mode, baud rate (if the serial board supports software controllable baud rate), protocol, "list status" (whether received files go to the printer or to a disk file) and the file name to be used in file transfers. All these factors can be changed during communications as well.

Table V
UARTS/USARTS Supported by MCALL

Intel 8251
Motorola MC6850
National INS8250
National INS8402
Signetics 2651
Texas Instruments TMS5501
Texas Instruments TMS6011
Various AY-(3,4,5)-(1013,1014,1015)

MCALL offers several protocols. The standard X-ON/X-OFF (Control-Q/Control-S) is normally used for communicating with time-sharing systems, but an alternate BREAK/RETURN protocol used mostly on Univac computers is also provided. MCALL uses its own protocol for transfers between personal computers. The protocol is based on similar concepts as that used by COMMX but has a few differences. A checksum system is used rather

than COMMX's CRC 16, which is simpler but not quite as sophisticated. An article written by Tim Pugh for *Dr. Dobbs* is included on the distribution disk; it is very well written and informative, teaching the reader about protocols in one painless lesson.

As with modem board versions of COMMX, the AMCALL version offers the choice of auto-answer or originate modes. An excellent enhancement is provided in originate mode. A disk file of commonly called numbers is maintained, and when originate mode is selected these numbers appear on the screen preceded by a single letter. Simply entering the chosen letter will dial up the system selected, establish connection and begin communications.

Once the user is satisfied with his initial configuration, a summary of the control characters support by MCALL is displayed (see Table VI). Note that escape characters are used to communicate with the program rather than normal control characters; this avoids problems with computers which intercept standard control characters and thus makes them available for sending to the connected system.

Table VI
MCALL Control Characters

ESC B	TRANSMIT A "BREAK"
ESC C	CLEAR BIG BUFFER (FILE RX MODE)
ESC D	DUPLEX MODE SELECTION
ESC E	EXIT CURRENT MODE
ESC F	FILE NAME SPECIFIED FOR SUBSEQUENT TX/RX
ESC H	HELP - DISPLAYS THIS COMMAND LIST
ESC L	LIST ON/OFF SWITCH (FILE RX MODE)
ESC P	PROTOCOL SELECTION
ESC R	RECEIVE (RX) A DISK FILE FROM A REMOTE DEVICE
ESC S	SIGNAL (BAUD) RATE SELECTION
ESC T	TRANSMIT (TX) A DISK FILE TO A REMOTE DEVICE
ESC W	WRITE BIG BUFFER TO DISK (FILE RX MODE)
ESC X	X'FER CONTROL TO DDT (ASSUMES PROGRAM LOADED BY DDT)
ESC ?	WHAT IS THE CURRENT SYSTEM CONFIGURATION?
ESC ESC	CONTROL CHARACTER DISPLAY ON/OFF SWITCH

Issuing any of the escape sequences pauses the communication process and clears the screen to display the command menu. The user can execute any sequence of commands and return to communications without losing the linkup. Several of the commands duplicate those in the configuration dialog which preceded connection; this allows parameters to be changed "on the fly" if the initial settings prove to be wrong.

ESC-B transmits a legitimate BREAK character to the host. This is necessary since CP/M does not support "break detect" and thus the BREAK keys found on many terminals will not work.

ESC-C works with ESC-W to provide complete control over the receiver buffer. A normal receipt of the file proceeds automatically, without operator intervention. The ESC-C/ESC-W mechanism allows the user to set up a receive file, talk with a remote system, and write to disk (or printer if the list flag is set) only those parts of the dialog worth saving. Thus MCALL has a more flexible logging system than COMMX since *either* complete dialogs or only selected portions may be retained for future reference.

ESC-^ toggles the display of control characters from the host on or off, while ESC-# allows the user to screen out any such control characters. This can be quite useful when first accessing a new host system; if strange things happen these commands can often isolate the problem, and sometimes solve it.

The AMCALL version offers one additional command. ESC-Z exits AMCALL and returns to CP/M without breaking the connection. AMCALL can be re-entered and communications resumed at any time. Regular MCALL also allows such exiting and restarting, but without a special command for that purpose.

Evaluation

COMMX and MCALL were tested in both serial board and modem board versions with an Industrial Micro Systems 440 I-O board and an Omnitec acoustic coupler and a borrowed PMMI modem board. Over a dozen CBBS systems, three mainframe hosts and several other personal computer systems were contacted. Files were transmitted in both directions, logs kept, etc.

Everyone likes clear-cut winners in confrontations, and I would love to say program A is clearly superior to program B, but this is simply not the case. BOTH programs worked perfectly. Any problems encountered with either when communicating with a remote system were always caused by inordinate line noise. This noise could sometimes be seen during normal terminal communications, but either program would make an accurate file transfer with another personal computer using the same software even under such adverse conditions (sometimes with quite a few retransmissions).

Thus the choice between COMMX and MCALL should be made on other considerations. COMMX comes preconfigured, is easy to use, and offers direct transfer of 8-bit files. MCALL is more flexible and has a larger command set, offers several features not available with COMMX and is even easier to use. Given that full source in either assembly language or C is available for about the same cost as the object code for COMMX, and MCALL is available for more systems, I expect that many purchasers would opt for either version of MCALL.

OS-1 — A Diamond in the Rough

David Fiedler

OS-1 is a disk operating system designed to run on minimally-configured Z-80 systems. To the user, it appears very similar to the popular UNIX system, incorporating many of its features. Since OS-1 isn't a multi-user or multi-tasking system, it does not need special hardware for bank-switching as do Cromix and uNIX (which are other Z-80 based UNIX look-alikes). A further attraction is the inclusion of a CP/M adapter, so that CP/M programs can be run under OS-1 control. OS-1 costs $249 (including one year's "software support") and is available from Software Labs, 735 Loma Verde, Palo Alto, CA 94303.

Bringing Up The System

If you have a CP/M 1.4 system, or can emulate one (apparently only the BIOS routines are used, and they expect strict conformance to standard single-density), bringing up OS-1 is simplicity itself. Just run the OS32 or OS48 program under CP/M (depending on your available memory), insert the eight inch OS-1 disk, and press any key. The OS-1 system doesn't care if you are running exactly a 32K or 48K CP/M system, as long as you have enough memory; however, any memory over 48K will not be used. No memory map is provided with OS-1, so it is not clear where everything goes. It just runs merrily along.

Three disks are included, one of which is in standard CP/M format, and contains the special OS boot programs mentioned above, the source code for certain parts of the OS-1 system, some library files and a loader. The other two disks contain a runnable OS-1 system and many utilities.

David Fiedler, Box 33, East Hanover, NJ 07936.

Understanding The System

Since OS-1 was modeled after UNIX, it comes as no surprise that the documentation also follows the UNIX format. The OS-1 User's Guide is 3/4 inch thick (the pages are not numbered in the normal fashion, again UNIX style) and describes every command, system call, and system concept, along with any particular files involved. It is an extremely impressive piece of work, and is less ambiguous than the corresponding UNIX manual in several places, especially in terms of the examples cited. Certainly it ranks as one of the most complete and well-organized software documents in the microcomputer industry. However, I do have some complaints. In consciously sticking to the UNIX guidelines, the authors sometimes get too attached to the UNIX terminology (e.g., talking about "dump tapes"), are unnecessarily terse, and can fall into the trap of giving overly general examples. Also, like UNIX, pure luck is necessary to find out certain minor, but important details—like at what address to assemble your files. (The answer is found in the section about the loader.)

An "Introduction to OS-1" was enclosed along with the OS-1 User's Guide. Although only 62 pages long, this manual proved an excellent beginning towards learning the essential concepts of OS-1. In fact, it is necessary to read this and the expository material in the User's Guide before you can make intelligent use of OS-1. This is difficult to do, as it is tempting to play around with the system before exercising your brain. I managed to survive this crisis, but only because I'm familiar with UNIX. I feel a very simple hand-holding guide is necessary for any system this powerful—something that would literally show you how to log in, tell you what to type in order to get certain results, and explain why you got the results. For

many people, it is not enough to describe what the commands do; it helps to be shown.

When OS-1 comes up, you are the "super-user" in the WIZARDS group (no, there is no ADVENTURE game on OS-1—yet). The super-user has all sorts of special access rights and powers, and can get into big trouble very easily. In particular, it is the super-user who has the capability to authorize new users, delete and edit special system files, and bypass virtually all file permissions. However, this power is very easily abused by a new user. My suggestion to Software Labs regarding this is to change things so that the beginner logs in as a normal user. You can still become the super-user at any time.

Backing Up The System

The disks I received were quite scratched on the recording surface, although the envelopes and outer jackets were in perfect condition. I suspect Software Labs is using some very old disk drives. I was able to read all three disks, although one was definitely close to being unsalvageable. I have found that banging the side of a disk drive while it is having trouble with particular disks will enable it to recover - sometimes.

Due to some peculiarities in my system which prevent me from using a "normal" single-density CP/M disk as a system disk, I was originally forced to pretend, as far as OS-1 was concerned, that I only had one disk drive. This also meant that I was unable to back up the distribution diskettes, since the two OS-1 disks are not in normal CP/M format. A sheet provided with the system advises the user to use a "raw disk utility" rather than CP/M's PIP to perform the backup, but none was provided, and I didn't have one that would let me back up on one drive. When I got the system up, I tried to use the OS-1 backup program. As a result, the system eventually crashed, leaving me with no working disks.

If you have a CP/M 1.4 system, or can emulate one, bringing up OS-1 is simplicity itself.

At this point, I was forced to call Software Labs. Without revealing my secret identity as a klutzy software reviewer for *Microsystems*, I explained what happened. The consensus was that I had run across a known (kr wn to them, anyway) bug in the cache routine that controls the updating of the disk. This made some sense in light of the particular way that the system crashed, and they agreed to fix up my disks and make backups for me in case I had more trouble. They suggested that I send the disks back, and they would return them as soon as possible, UPS Blue Label, C.O.D. To prevent even further delay, I sent $6 to prepay the postage. Then I waited for the swift return of the disks.

And I Waited...

Five weeks later, I received the disks, the backups they made, and a letter which implied that it had not been the cache routine bug that had laid my disks low. They did not mention whether or not the bug had been fixed. Unfortunately, (this is not completely clear in the documentation) the backup program cannot be used to back up only one disk drive, even though the program prompts lead you to believe this can be done. Therefore, it isn't possible to back up OS-1 on a single-drive system with the programs provided. It was disheartening, also, to find out it had taken five weeks to just copy a few disks.

In fact, it is not possible to run OS-1 on an unmodified CP/M 2.0 or later, which includes MP/M also. The reason lies in the sector blocking/deblocking algorithms present in these new versions, which permit certain disk write operations to take place without actually writing to the disk. What must be done involves patching your BIOS to fool it into thinking that all disks writes are TYPE 1 (write to directory sector—see page 34 of your "CP/M 2.0 Alteration Guide"). If this means little or nothing to you, there will be much grief before you ever get OS-1 running! After discovering this remedy, however, I was able to run OS-1 with both disks, and without crashes.

Running The System

OS-1 generally runs as advertised. After using CP/M on your machine, it is nice to see a system that "learns" what you are doing, buffering your most recently used data in RAM to avoid extra disk accesses. It is possible to run several commands sequentially in this way, and (of course) the response time improves greatly when the system knows about the buffering. Some of the programs seem to take longer to run than their size would indicate; this is probably because they have to traverse OS-1's tree-like file structure. It is hard to reconcile certain utilities' size with their functions—for instance, PIP is only 3.5K, while STTY (which simply sets up certain modes for the console terminal) takes up 13K. I suspect that some of the larger programs were written in C or Fortran, rather than assembler.

The system indeed looks like UNIX. The chief differences are in some of the special characters used:

	UNIX	OS-1
Prompt	$	-=>
Directory	/	:
Argument or Switch	-	/

So a typical command line, asking for a "long" or "detailed" directory listing would look like this under UNIX:

```
$ ls -l /usr/src/cmd
```

while the equivalent command for OS-1 would be:

```
-=> list :usr:src:cmd: /d
```

Although Software Labs feels it adds identity to OS-1 by using this syntax, I personally think it could avoid confusion if they went along with the UNIX conventions, perhaps by making the characters in question could be made user-modifiable. In any case, redirection of input and output to any device or file is supported, as is pipelining between programs (done by using temporary files). These features, along with the hierarchical file structure, are among the most useful and desirable in an operating system.

There are 68 commands associated with OS-1, comprised of 28 intrinsic commands built into the system (as opposed to five in CP/M 1.4), and 56 executable program files. Several reasons exist for this disparity:

1. Certain programs (the commands line interpreter, or "shell"; the debugger; and the CP/M adapter) exist as three separate files, and the appropriate one is executed depending on available memory. This decision, however, must be made by "linking" the name of the file you wish to execute to the correct size program for your system. As such, this is not even mentioned in the manual, and can leave you with the impression that you have much less memory available than is actually the case--especially with the CP/M adapter. It would be better if these files were relocatable, so you would only need one. Also, "login" appears both as an executable file and as an intrinsic command, implying that the file portion is needed to execute.

2. Some programs listed in the manual are not included on the disks (i.e., find and lead (a UNIX-type editor)). These programs I would classify as "missing."

3. Some programs included on the disks are not listed in the manual. Aget, aput, bget and bput seem to be command files (similar to .SUB files on CP/M) for transferring data between CP/M and OS-1. If this is truly their function, a little explanation would be helpful. It is not clear whether they were ever meant to be included, as they reference directories that do not exist on the distribution disks. Also, a bit of imagination is necessary to discover that the documentation for "sdcheck" (small directory checker?) is included with "dcheck."

Getting Annoyed With The System

The trouble with the large number of utilities is that it's difficult to exercise them properly. Both disks are rather full, and you quickly run out of i-nodes (somewhat akin to directory space in CP/M) when attempting to add users, test files, or the like. While the "init" program is all that you need to create a new usable disk for OS-1, I couldn't discover how to make a new system disk that would boot. I was unable, therefore, to put together a set of disks that had enough room, and utilities, to do what might be considered "useful work." This was made more difficult by the condition of the utilities themselves, as well as the warning Software Labs includes, "files currently included will not be compatible with later equivalent editions." In fact, the "loader will not load files produced by itself." At least they warn you!

In the process of looking through the various files, printing out listings, logging on, and being super-user, I noticed funny things happening. An entry for a new user would be created; it could not be deleted. The rmuser (remove user) program for doing this would not work properly, so I finally gave up and edited the user file directly. The system, however, prevented me from writing on this file, even though I was the super-user and supposedly could bypass all file permissions. So I tried to test the disk—perhaps there was a bad sector. Running the "badblk" program, with the /1 switch set as shown in the manual, would list out the bad areas of the disk. Instead I was told I had a bad switch. This same sort of problem cropped up in various other utilities, and I was continually frustrated when things didn't work as they were shown in the manual. Sometimes the switches in a given program would respond to upper case, even when lower case was shown in the manual.

Certainly the many prompts are helpful in telling you where you are going wrong (with the possible exception of "That command did nothing"). But the problems faced in trying to work with OS-1 have, for me, outweighed the nice features. The system seems to be carefully thought out, yet the implementation is lacking.

In a phone conversation with John White, president of Software Labs, I was asked to note in the review that OS-1 was purchased from the original owners (a company called Electrolabs, now out of business) and is currently being supported completely by Software Labs. I was told that Software Labs was writing their own C compiler, that OS-2 (multi-tasking) was coming along, and that they were working on OS-3 for 16-bit machines. Also, the new update for OS-1 was on schedule, and in fact he had the first draft on his desk.

Over three months later, I have not heard anything about the promised update, the compiler, or even the cache bug fix. I believe the people at Software Labs have the capacity to develop fine products, but it sounds to me as if they are moving ahead before they finish their past commitments. OS-1 has not been finished yet, and I can't recommend it until it has been.

Trademarks: OS-1 is a trademark of Software Labs. UNIX is a trademark of Bell Laboratories. Cromix is a trademark of Cromemco. uNix is a trademark of Morrow Designs.

The BDS C Compiler

David Fiedler

In the past few years, there has been a great deal of interest in the C programming language. Proponents claim C is everything Pascal should have been, critics sneer that its syntax is too cryptic (and sometimes call it "C" with quotation marks, as if it weren't quite legitimate), and software houses write compilers for it.

C will always be associated with the UNIX operating system, because one of UNIX's special features is that it is easily maintainable. It is easily maintainable because it is written in C. A slightly less well known fact is that C was developed to be a suitable language in which to write UNIX. C is, therefore, a high-level language capable of the efficiency necessary for writing serious system program. It does not have either the strong type-checking or the addressing constraints of Pascal. Programs written in C are highly portable between machines having C compilers. All this makes C just the kind of language useful to serious microcomputer users—that's how C ended up running under CP/M.

There are three C compilers that have been generally available to the CP/M user:

Small-C	$ 15 by Ron Cain
BDS C	$150 by Leor Zolman, BD Software
Whitesmiths C	$600 by Whitesmiths, Ltd.

These compilers represent a wide range of price and capability. This software review deals mainly with the "mid-priced" model, the BD Software C compiler (BDS C), comparing it to the other two when appropriate.

In almost every field of endeavor where a purchase is involved, the question, "Which should I get?" comes up. The answer, as usual, is "What do you want to do with it?" If you just like to collect compilers, or enjoy fooling with them, then Small-C is a bargain. The Small-C compiler is completely written in Small-C, and the source code for both the compiler and the run-time library is included for $15. If you absolutely must have a compiler with the full language capabilities of Version 7 UNIX, then only the Whitesmiths compiler will satisfy you.

However, if you are looking for an affordable compiler that is easy to use, produces respectable code, is usable for systems programming, and is supported by an active users' group, you should take a serious look at BDS C.

The "full language" clause above deserves some explanation. Unlike other popular languages such as Basic, there is a published standard for C (Appendix A of *The C Programming Language*, by Kernighan and Ritchie, Prentice-Hall, 1978). So it is easy to determine the extent to which a given C compiler supports the language. But, similar to Pascal, the language definition does not mention I/O facilities (though a standard I/O library is defined by Dennis Ritchie in the UNIX documentation). Since most C programs need to do some kind of I/O, portability suffers if the supplied I/O library does not follow the standard.

If you absolutely must have a compiler with the full language capabilities of Version 7 UNIX, then only the Whitesmiths compiler will satisfy you.

What does all this mean? Let's take the archetypal C program, the first one mentioned in the book:

```
main()
{
    printf("Hello, world\n");
}
```

Of the three compilers studied, this program will only compile "as is" on BDS C, due to differences in the supplied I/O libraries. Similarly, differences in names and calling sequences of other library functions prevent typical (i.e. copied out of Kernighan and Ritchie) C programs from being compiled on Whitesmiths or Small-C without a certain amount of editing and rewriting. While BDS C is not totally free from these restrictions, it is more amenable to running programs straight from the book.

This is extremely important because it is impossible to learn C without writing and running programs in it, and it helps to start with the examples in the book.

Documentation

The Kernighan and Ritchie book is best for introduction and reference, but still does not serve all tutorial needs. So the compiler manual will become your constant companion.

BDS C comes with a 72 page user's manual written in a clear, personal style. There is no index or table of contents, but the manual is separated into logical sections. What the manual lacks in formality, it makes up for in hints, short tutorials, program examples, and pithy comments on software in general. Leor went to the trouble of getting the entire manual typeset in genuine Bell Labs style, which helps readability greatly. You are expected to be familiar with CP/M, C, and things like hex and bytes—but this knowledge is not necessary to use the compiler.

Small-C comes with an eleven-page user's guide, which is more than adequate. It clearly explains how the function calls work, what the compiler's limitations are, and how to work the compiler itself. It's all you really need to sit down and get started—one part walks you through a compile-and-load session. No index here either, it's really not necessary.

Whitesmiths C comes with two 1/4 inch thick reference manuals. It's hard to describe the style of writing; the word "terse" does not quite do it justice. The authors are fond of using their own definitions to help explain how the functions are supposed to work, a habit which can confuse the reader.

The explanations of almost every subject are included, but they are often hard to find, extremely cryptic once found, and sometimes inconsistent. An index is absolutely necessary, but only a totally inadequate (I feel page numbers would help greatly) table of contents is included. These manuals are completely lacking in readability, and detract from the product as a whole.

General Comments on BDS C

One reason for BDS C's speed of compilation is that, unlike the other C compilers, it does not produce an assembly code directly. This could be a factor if you must trace through program execution; you won't have a listing to guide you. Generally, though, the code generated is straightforward, and can be followed with DDT. If you have Digital Research's SID debugger, you can use a loader option under BDS C to write out a symbol table to disk. This aids debugging. The loader also will allow you to create ROMable code, and overlay segments. This means that you can write programs bigger than available memory, and bring extra segments in when needed.

Most of the limitations of BDS C are due to its restricted subset of the language, and to the way CP/M operates.

Functions are stored in a relocatable format, and are kept together in larger "library" files. An interactive librarian program is included, which lets you create new function libraries and change old ones. When I found a bug in one of the library routines, this let me simply replace the old one with the new improved version. There is a section of the user's manual about the format of these relocatable files, for those who wish to write their own in assembly language, and a macro library to help even further. Of course, any C function can become part of the library by just compiling it, and using the librarian.

The compiler does some optimization (for speed or size—your choice). Also, if you are using a Z80 processor, this fact will be automatically sensed, and block move instructions will be used when appropriate. I have found that BDS C generally produces the fastest code of the three compilers.

If you write programs for sale using their compiler, Whitesmiths will expect you to pay them royalties because you will be distributing parts of their run-time and function libraries along with their code. Leor Zolman has specifically declared that none of this sort of thing is necessary with the BDS C compiler. And the entire Small-C compiler is in the public domain.

Nobody Is Perfect

Most of the limitations of DBS C are due to its restricted subset of the language, and to the way CP/M operates.

BDS C does not support float, long or double types; static or register storage classes; initializers; blocks, or parameterized defines. Externals are handled like COMMON in FORTRAN; there are external variables, and no external keyword. There are other differences covered in the manual, but these are the major ones.

What About Tiny-C?

Conspicuously absent from this issue is a review of the Tiny-C Two compiler from Tiny-C. The company had furnished us with a review copy of their package and we arranged for the review with an individual who very regretfully failed in his committment. We then asked Tiny-C to supply a second copy for another reviewer who still has not completed his review. We're hopeful that we will be able to furnish you with this review in the near future. The Tiny-C people certainly have been cooperative in supporting our reviewing efforts, and we feel most guilty since they were the first to supply a review copy of a C compiler, and will be the last to be reviewed.

I would therefore like to point out that Tiny-C Two does support the redirection features of Unix C which are missing from the Small-C and BDS-C compilers. Further, it supports 32-bit integers (compared to only seven in BDS-C) and hence is viable for scientific and business applications. Also, the full source code is included with the $250 price (manual alone is $50) and a 20% reduction is given to owners of the older Tiny-C One interpreter. *—Editor*

How important are these missing features? It depends on the type of the work you plan to do. While the lack of floating point might lead you to assume that you couldn't write *any* business or scientific programs, this is not quite the case. A floating point package is included with BDS C, which lets you work to approximately seven digits of precision. It does work and is not terribly hard to use, but it is rather slow. You wouldn't want BDS C if you intend to write an Accounts Receivable system.

Initializers are a convenience only; several functions included let you initialize variables in the code. External storage can often be used instead of static, and there is now a "long" package available from the BDS C User's Group. Generally, long variables and static storage are the only things you will really ever miss for system-type programming.

You might still be wondering if there would be a limit on your creativity. Currently, you can obtain the following types of programs, all written in BDS C:

- Adventure
- 6800 and 1802 Cross-Assemblers
- A text processor
- A file comparison utility
- Extended Directory Listing
- The Game of Life
- A disk "zapping" program
- A full-screen text editor
- An intelligent file transfer and communications package

One of the attractions of Whitesmiths' product is that redirection to/from files and devices is supported as on UNIX.

With the exception of the last two products mentioned (which are commercially sold under the names MINCE and AMCALL), these useful programs, and others like them, are available free from the BDS C Users' Group. And apart from the editor and the Life game, none of the programs appear to run slower than an equivalent assembly language version might. The point is that BDS C provides a viable subset of the C language.

While not a limitation of BDS C, it should be noted that CP/M is not exactly like UNIX. On UNIX, you can redirect input so that a program which expected input from the console could get that input from a file instead, without changing the program. Similar redirection is possible for output, in fact, any device could be treated like a file for this purpose. CP/M, of course, doesn't have these features. For this reason, BDS C programs which call the **getchar** function (used frequently on UNIX for reading files as above) will get their input from the console only. File I/O must be done deliberately. Small-C works in a similar manner.

One of the attractions of Whitesmiths product is that redirection to/from files and devices is supported as on UNIX. This is done in their run-time I/O library, and is one reason why programs written in Whitesmiths C tend to be somewhat large. The ideal situation is to have this facility in the operating system.

Do You Want to Own It, or Just Use It?

In my opinion, the usefulness of a piece of software is in part measured by its speed and size. If you intend to use a program often—an editor or compiler, for example—it will become a burden if the time needed to use it is too great. I compiled the simple C program listed above with all three compilers (and on UNIX just for comparison), making changes as necessary. The results are listed in Table 1.

Table 1.

COMPILER	minutes to compile, link, and load	actual code generated (bytes)	executable program size	time to execute
BDS C	0:25.1	34	2304	2.8
Small-C	1:42.8	44	3200	4.2
Whitesmiths	4:04.1	28	15232	5.1
UNIX (20 users)	1:02	20	4184	1.0
UNIX (optimized)	1:25	20	2468	1.0
UNIX (single user)	0:17	20	4184	1.0

The intent of testing such a simple program was to get the minimum times and sizes (due to the size of the run-time package) possible for each compiler. Be assured that compile times can increase considerably for even slightly larger programs. In the case of BDS C, the entire compile and link process was faster than the assemble and load alone for Small-C (done by the standard CP/M utilities). This means that BDS C can give you executable code more quickly than if you wrote the equivalent program in assembler. And it is usually much faster to write in C than in assembler.

While repeated compilations are no substitute for careful program design, it is maddening to wait many long minutes, only to find that you made a slight error. Then you have to edit and recompile. I first began programming in C with the Whitesmiths compiler and much slower disks. The frustration level was so high that I gave up C altogether for several months. Using this compiler also necessitated doing all sorts of hardware modifications to my system, so I could make ROMs disappear and have enough memory to run it. Even now, I am looking for a way to make my Zapple monitor respond to the PHANTOM bus signal, so I can squeeze out another 2K of RAM.

When I got Small-C, I was favorably impressed by its speed and error reporting, but the severely limited subset of the language and I/O facilities frustrated me again. The simplest programs available had to be completely reworked to fit the Small-C format, and I found myself spending more time working around the compiler than using it.

BDS C changed the whole picture. I was able to write programs the way I wanted, getting fast results. When I made a mistake, the compiler was able to show me how

and where, in unambiguous language. Within days, I felt comfortable enough with the documentation provided to contact Leor Zolman telling him I thought I had found an obscure bug in a library routine. By the time he had called me back with the fix, I was able to fix it myself, and install the new version in the function library. This was only a matter of an hour or so. The importance of all this? I could do these things because the entire function library is included on disk in C source code form. Not only that, but the run-time library is also included (written in 8080 assembly language). This is invaluable material for learning more about how the functions work, as well as having a lot of good C code to study.

Bugs? After looking through issues of *Lifelines*, which lists bug reports and new versions of both BDS C and Whitesmiths, I found that Whitesmiths tends to release versions at longer intervals. Both compilers had serious bugs reported (crashes, incorrect code, etc.). This means that you would have longer to wait for a fix if you own Whitesmiths.

Now, what about updates? Since you get the source for much of what is important in the BDS package, it is possible, in many cases, to get bug fixes in source form as I did. And disk updates for the whole package are available for $8 from the BDS C Users' Group.

In the case of Whitesmiths, things are a bit more complicated. I got an early version of their compiler (1.1), and received one "free" update, for which I had to pay a $30 "media charge." Now, as the proud owner of Version 1.2, I find in *Lifelines* that this version has such a multitude of serious bugs that immediate upgrade to 2.0 is recommended. The only problem is that an update from Whitesmiths would cost me a $200 fee (plus another $30 media charge).

Final Words

A good analogy for these compilers might be to compare them to items with similar price relationships:

For $15 or so, you can buy a cheap camera, and take properly exposed, slightly fuzzy pictures at an average distance. You don't have to be very committed to photography to be able to use or afford one of these.

For maybe $150, you can find a high-quality 35mm SLR camera, which can take professional quality pictures. It will be more versatile, and will give you a great deal of enjoyment. While you don't have to use it to potential, it is nice to know that the user will almost always be the limiting factor in its performance.

Finally, you can spend $600 and get one of the most well-known and respected models, the kind all the "pros" use. You can impress your friends with how much it cost, and you might just take some good pictures with it. Try to take good care of it, for if something goes wrong, it will probably cost more to fix than the purchase price of the $150 camera. It might be a bit harder to use, and have its own peculiar problems, but that can be expected—it's special, remember?

Unless you must have floating point and statics (and have a very fast disk system with lots of RAM), I suggest you buy BDS C if you want to use a C compiler on CP/M.

Notes

1. All CP/M times were obtained on a 59K 3 MHz Z-80 based system running double-sided, double-density 8" floppies.
2. Times for BDS C were done on a single-density disk, and so are probably 20% higher than they would be otherwise.
3. Whitesmiths compile alone (all three passes) took about 1:30. The rest was spent in loading (this loader is notoriously slow). Note that large size of run-time package reflects additional UNIX-like capabilities.
4. Sizes of executable code (.COM files) were calculated by the number of records, so each could be as much as 127 bytes too high.
5. The string being printed takes up 14 bytes itself, which is included in the totals for actual code generated.
6. UNIX times were obtained on a PDP-11/70 running UNIX Version 7, with two 176 MB hard disks. The ideal home system.
7. PDP-11 code was used for code sizes on UNIX. Notice also the difference in compile times between average (20 users) and no (1 user) system loading.

UNIX is a trademark of Bell Laboratories.
PDP-11 is a trademark of Digital Equipment Corp.

Where To Find Them

BDS C Users' Group
409 E. Kansas
Yates Center, KS 66783

Membership is $10/year domestic, $20/year foreign. This brings you regular newsletters. You don't have to join to buy disks, at $8 apiece for 5 1/4" or 8" size. In the future this group hopes to support other C compilers but currently they only support BDS C.

Whitesmiths, Ltd.
P.O. Box 1132
Ansonia Station
New York, NY 10023
(212) 799-1200

The C compiler requires 60K CP/M at a minimum. Version 1.2 was reviewed.

(source for BDS C)
Lifeboat Associates
1651 Third Avenue
New York, NY 10028
(212) 860-0300

Version 1.43 of BDS C was used for this review. System size: 32K CP/M minimum, 48K recommended.

Lifeboat distributes BDS C and Whitesmiths. The update fees may be a bit better than directly from Whitesmiths, and they sell software in almost any format you can name.

The Code Works
P.O. Box 550
Goleta, CA 93017

Version N was used. No recommended system sizes are given, but the compiler itself takes 22K, and it was developed on a 40K system with a single mini-floppy.

An Introduction to the C Programming Language — Part 1

David A. Gewirtz

An introduction to the C language and reviews of four popular microcomputer implementations

C is the language that was developed after B. Although its name isn't very original, the C language is a highly versatile programming tool. C cannot be classified either as a "higher" or a "lower" level language. Probably the best classification would be a low-level, higher level language. It has also been called a systems implementation language.

C has many of the higher level control constructs, such as while loops, if-else conditionals, and block structuring. C also easily manipulates machine-related data types such as the byte with operations like bit-wise shift This combination makes C very useful for programming operating systems, languages, and utilities. It can also be easily used for other applications as well.

C, originally developed at Bell Laboratories, has evolved from the BCPL language created by Martin Richards, and the B language written by Ken Thompson for the first UNIX operating system. It shares many of the features of both BCPL and B, although where they concentrated almost exclusively on machine words, C has been expanded to include larger integers, characters, and in some implementations, floating point numbers. In addition, the syntax is different between BCPL and C. C also borrows many ideas, like the typing of variables, from other languages including ALGOL, Pascal, and PL/I. The most noted application of C was the development of the UNIX operating system for the PDP-11 computer.

Unlike Basic, C is a compiler, not an interpreter. This means that once compiled, programs run much faster because the statements in the program do not have to be retranslated into machine code each time they are encountered, which can become a very time consuming operation. In addition, the resulting object program that is run takes up much less space because the interpreter does not have to be resident in memory while the program is running.

C does not have the line numbers used in most versions of Basic. Instead, a C program is organized into manageable routines called functions. These functions, combined with the ability to make compound statements, make programs very straightforward. Wherever a simple statement can be used, a compound statement can be used there instead. For example, in the following "b=9" is a simple statement:

```
if (a==5) b=9;
```

Any group of statements, enclosed in brackets " " and " ", can be substituted for that simple statement:

```
if (a==5)
      {
      b=9;
      c=36;
      if (count<8) ++count;
      }
```

The operators "==" and "++" will be explained later.

These compound statements have a recursive definition. A compound statement is a statement containing other statements. Any of the statements inside a compound statement can also be a compound statement. This very powerful way of using program control statements allows programs to be "block-structured." I will not argue the relative merits of structuring, save that it makes programs easier to design, and clearer to debug and understand. This same block-structuring construct can be found in ALGOL and Pascal.

Functions are usually small, stand-alone subprogram segments of the main program. They are similar to the

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

Basic gosub statement in that when they are called, control jumps to them, and at the end they can return to where they came from. That is the only similarity. In Basic, all variables are global. If a routine to position the cursor on a terminal at X and Y coordinates was used in Basic, it might look like this:

```
330 REM Position Cursorat A and B
340 X=A
350 Y=B
360 GOSUB 5000

4999 REM Direct Cursor Addressing
5000 PRINT CHR$(27)+"="+CHR$(32+Y)
     +CHR$(32+X);
5010 RETURN
```

You would not be able to use X and Y for anything but that routine, and would have to assign the actual values A and B each time the routine had to be called. In addition, you would have to remember that 5000 was the cursoring routine. The same thing in C would be the call:

```
cursor(a,b);
```

which is much clearer to understand. The function would be:

```
cursor(x,y)
      {
      outchr(27);
      outchr('=');
      outchr(32+y);
      outchr(32+x);
      }
```

Outchr is another function which puts one byte to the screen. Once the cursor function is known to be working, it can be put aside and ignored, save to remember to use the function cursor(x,y) whenever it might be needed.

C makes very extensive use of the function feature. The C compiler can be relatively small in terms of reserved compiler operations, and all extraneous operations can be custom designed to fit a user's needs. In fact, C does not have any defined I/O, rather it consists of machine language functions to be called when needed. With functions, programs are very simple to modify. For example, if a new terminal were to be used, all that would be required would be a change in the function. The program remains the same.

Most C compilers come with a run-time library that consists of many previously compiled standardized functions such as outchr and cursor. The programmer simply has to remember the rules of what goes in and comes out of each function, but doesn't even have to include them in the actual program file. They simply must be "linked" to the compiled program before it can run.

The flow of control throughout C programs also helps to promote the readability and structure of the programs. C has many statements that evaluate conditions and control program flow accordingly.

if (expression) < statement > [else < statement >]

If a given condition is true, do something (a statement which can always be a compound statement), else (if it is not true) do something else. If the else is omitted, flow will continue directly after the statement with the if.

while (expression) < statement >

While a given condition is true, do something until that condition is no longer true.

do < statement > while (expression)

Do something while a given condition is true. This differs from the while in that the condition is checked after running through it, not before.

for (expr1; expr2; expr3) < statement >

This works somewhat like the Basic FOR statement. It controls looping through a statement by evaluating three expressions. The first expression is evaluated once, then as long as expr2 evaluates to something other than zero, the statement is executed and expr3 is evaluated. This could be seen as:

```
for (i=n ; i!=j/3 ; ++i) <statement>
```

which says "for I becomes n, until i equals j/3, do the statement, and increment i." The symbol "++" is an increment operator which will be examined later.

goto < identifier >

Branch to the label specified by < identifier >.

switch (expression) < statement >

This can be seen as a very powerful ON-GOTO statement. It evaluates the expression, and if it matches any of the cases in the statement, it will execute from that point. A break statement will return control outside of the block. For example:

```
printf("What is your choice?");
getchr(choice);
```

This will get the choice for some set of options. Toupper is a function to convert lowercase alphabetic characters to their uppercase equivalent. If "B" was chosen, the function called reboot would be executed. When it finished, the break would be encountered which would return control outside the switch. If "Z" were chosen, the screen would be cleared, and if none of the options listed were chosen, the error message would be printed.

In addition to the statements above, C has a few very powerful operators. Unary operators are operators that do an operation to a single piece of data. The following is a list of the C unary operators. All of the operators return a value to some expression. For example, in Basic, -X does not mean negate X, but rather returns the negative of X to a specific expression.

```
*p - Pointer to nnn.
     The value is the value of the object currently
     pointed to by nnn.

&x - Address of x.
     Returns a pointer to x.

+x - States that x is positive.
     Returns the same value of x.

-x - Negative of x.
     Returns the negative of x.

++x - Increment x.
      The result is the new value of x.

--x - Decrement x.
      The result is the new value of x.

x++ - Increment x.
      The result is the original value of x.

x-- - Decrement x.
      The result is the original value of x.
```

```
~x - Returns the ones complement of x.

!x - Not x.
     The result is 1 if x is 0, otherwise 0.

(type-name)x - coerce x to the type type-name.
     Returns the value obtained by converting the
     value of x to that type.

sizeof x - The result is an integer value equal
     to the size in bytes of x.

sizeof (type-name) - The result is an integer value
     equal to the size in bytes of an object of type
     type-name.
```

There are also quite a few binary operators available for C:

```
x*y - Multiply.
      The result is the product of x and y.

x/y - Divide.
      The result is the quotient of x divided by y.

x%y - Remainder.
      The result is the remainder of x/y.

x+y - Add.
      The result is the sum of x and y.

x-y - Subtract.
      The result is the difference of y from x.

x<<y - Shift left.
       The result is x shifted left y bits.

x>>y - Shift right.
       The result is x shifted right y bits.

x<y - Less than comparison.
      The result is 1 if true, 0 otherwise.

x>y - Greater than comparison.
      The result is 1 if true, 0 otherwise.

x<=y - Less than or equal comparison.
       The result is 1 if true, 0 otherwise.

x>=y - Greater than or equal comparison.
       The result is 1 if true, 0 otherwise.

x==y - Equal to comparison.
       The result is 1 if true, 0 otherwise.

x!=y - Not equal to comparison.
       The result is 1 if true, 0 otherwise.

x&y - And.
      The result is the bit-wise and of x and y.

x^y - Exclusive or.
      The result is the bit-wise exclusive or of x
      and y. Be careful not to confuse this with the
      "power of" facility of Basic.

x|y - Inclusive or.
      The result is the bit-wise inclusive or of x
      and y.

x&&y - Logical connective and.
       The result is 0 if x is zero, without
       evaluating y. Otherwise the result is 1 only
       if both x and y are non-zero. This is used in
       logical expressions such as:

  if (a==b && b==c) a=c;

Evaluation stops as soon as the expression
       becomes false.

x|| y - Logical connective or.
        The result is 1 if either x or y is non-zero.
        This is used in expressions such as:

        while (a!=b    b!=c    d!=q) ++a;

t?x:y - Conditional evaluation.
        This acts like the statement:

        if (t)
              {
              x;
              }
        else
              {
              y;
              }

The result is the result of the evaluation.

x=y - Assignment.
      The result (which is x) is the value of y.

x*=y - Equivalent to x = x * y.

x/=y - Equivalent to x = x / y.

x%=y - Equivalent to x = x % y.

x+=y - Equivalent to x = x + y.

x-=y - Equivalent to x = x - y.

x<<=y - Equivalent to x = x << y.

x>>=y - Equivalent to x = x >> y.

x&=y - Equivalent to x = x & y.

x^=y - Equivalent to x = x ^ y.

x|=y - Equivalent to x = x | y.

x,y - x is evaluated first, then y.
      The result is the value of y after evaluation.
```

Although these operators make writing programs much easier and less verbose, their conciseness tends to make the programs difficult to read.

Variable and function names can generally be of any length, and use any displayable characters, as long as they do not conflict with the reserved words. Data in C is organized in both types and classes. A storage class is a method of storing the data. C has the following storage classes:

extern — Specifies that an external definition for the given identifier outside of that file.

auto — Specifies that the given identifier will exist as dynamic local variable. It will be created when that block of code is executed, and destroyed when the block is exited.

static — Specifies that the given identifier will not be known outside a block, but will remain inside that block. It differs from auto in that the data declared static will always exist until the termination of the program. This is similar to Algol's "own" declaration.

register — Means the same as auto, but specifies that efficient storage, such as registers be favored to hold the object, and the address of the object (&x) cannot be taken.

typedef — Specifies that the identifier should be recognized as a type specifier. This is used to create new data types from old ones. This is similar to Pascal's "type" declaration.

Data types are relatively primitive in C. Arrays are constructed of scalars; many implementations do not have them at all. The following is a list of data types that might be available:

char — A byte integer used to hold a single character of the machine's character set.

int — Usually a two byte integer, can also be called "short" in some versions.

long — Usually a four byte integer.

float — A floating point number, usually four bytes.

double — A double precision floating point number usually eight bytes.

struct — A sequence of one or more types used to create logical records in a program. Similar to Pascal's "record" and COBOL's record. In some implementations can be used to access individual bits in a byte as a field.

union — A record specified by fields.

pointer to — An unsigned integer used to hold an address of some object.

```
char line[80], *pline;
```

*pline is the pointer to declaration.

array of — A repetition of some type. Example is above in line 80 declares 80 repetitions of type char, and aligned it on line.

Pointers provide a very powerful and sometimes confusing tool in program writing. Simply stated, a pointer points to some location in the machine. They can always point to any location in the machine's address space. A pointer can be manipulated, or by placing an asterisk in front of the name, that which is being pointed to can be manipulated. For example:

```
char fcb[12], *pfcb;
```

Declares an array of 12 bytes with the first byte called fcb. If fcb is located at 54A2h, then the following would set the pointer pfcb to 54A2h:

```
pfcb=&fcb;
```

Note the value of pfcb has been changed to 54A2h. If we wanted to increment the pointer to point to the second byte, the following could then be done:

```
++pfcb;
```

Now, if we want to set what is now at 54A3h to "?", then the following could be done using indirection through pointers:

```
*pfcb='?';
```

This is where C can be confusing:

```
*++pfcb='?';
```

does the same thing. It increments pfcb to 54A3h and puts a "?" at that address.

C is not perfect however, there are some major disadvantages as well. One is the problem of operators being intermixed in confusing ways, as with *++pfcb.

Another rather serious problem is in the definition of the language itself. There is no actual formal definition of the C programming language. The closest is in a book written by the designers of the language [Kernighan 1978] in which there are quite a few vague points where some things are not absolutely explicit. Because of this, some implementations may differ in points of interpretation.

The next problem is that of portability. Although many implementations claim to be transportable to systems using compilers other than their own implementations, such as UNIX systems, they may in actuality be transportable to only a few other systems. This is not because of the syntax of the language, which will probably be identical, but because of the definition of the functions used. It may be necessary to rewrite some of the functions used to do what you expect them to do.

Lastly, and this is a mixed blessing, the C compiler is not overly picky on the whole. It will allow types to be confused without doing anything about it, and this can lead to difficult code to read, as well as results that may be CPU dependent. On the other hand, if you want to do something fancy on a specific machine, it will allow it.

Although there are drawbacks, C is a very powerful higher level language that is simple to use. It is one of the best for doing any form of systems software, and has quite a large user following.

Name: BD Software C Compiler
Price: $145
Distributed by:
Lifeboat Associates
1651 Third Avenue
New York, NY 10028

The BDS C Compiler was written by Leor Zolman of Cambridge, MA, and is distributed by Lifeboat Associates in New York City. This compiler takes the large step from a small development language to a full scale production compiler. Although not nearly as large or complex as compilers residing on mainframe computers, the BDS C compiler can handle a significant portion of the programming requirements of a microprocessor-based small computer.

Unlike Basic, it is not an interpreter but an actual compiler that generates machine code. Unlike most Pascals, it is a native code compiler. That is, it generates machine code directly executable on the machine for which it was designed (in this case the 8080, Z-80, and 8085 microprocessors) without generating an intermediate source code such as assembler source code or Pascal "P-code".

The BDS C package comes with a CP/M-compatible floppy disk containing a version of the compiler (in this case version 1.43) and the linker. In addition, all of the support modules containing compiled or assembled functions, as well as the run-time package, are also included. Although the compiler itself is not furnished in source form, the C run-time package and the two standard function libraries written in C, and the standard assembly language function libraries are furnished in ready-to-run versions and as source files. Since any C program is (or should be) made up of many separate functions, the BDS C package contains a library manager called CLIB that allows the programmer to build library files out of compiled functions from many files.

Although the BDS C Compiler does not directly support floating point numbers, a set of functions is included that allow floating point numbers to be manipulated in a reasonable manner when necessary. Another set of functions allows the programmer to integrate assembler code into callable functions. These routines require the assistance of the Digital Research Macro Assembler however, which is not included on the disk. Lastly, a

number of example programs are included, such as a working Othello and a rather nice terminal emulation program called "Telnet." Once Telnet has been customized for a particular computer, it provides intelligent terminal operations such as file transfer and receive. With another computer also running Telnet, it will transfer compiled files as well. I have run Telnet for quite some time to provide communication between my computer's acoustic coupler and a PDP-2060, as well as a PDP-1050, and Micronet.

In addition to the floppy disk software and documentation, BDS C comes with a seventy page manual that describes the BDS implementation, how it varies from previous BDS C versions, how to use the compiler, the linker, and the librarian. Also listed is usage documentation of the user available functions. Finally, comparision of BDS C to the "standard" C specified in Kernighan and Ritchie is also included.

Included in the BDS C package is a book by the authors of the original C language from Bell Labs, *The C Programming Language*, by Brian W. Kernighan and Dennis M. Ritchie. If anyone seriously intends to program the authoritative reference on C and is a necessary item in any C programmer's library.

In addition to the floppy disk software and documentation, BDS C comes with a seventy page manual that describes the BDS implementation, how it varies from previous BDS C versions, how to use the compiler, the linker, and the librarian.

BDS C complies with the syntax specified in Kernighan and Ritchie and supports many of the features of C—with some exceptions. These include short int, long int, float and double. Also there are no explicitly declared storage classes. Static and register classes do not exist; all others are either external or automatic. That is, if a function is called and it is not in that file, it is simply left as a reference to be resolved by the linker. Initializers are not allowed, so you cannot declare a variable and assign it a value in the declaration section. Unary operators "(type-name) expression" and "sizeof (type-name)" are not implemented. In structure and union declarations, bit fields are not implemented.

However, the preprocessor functions #define, #include, #ifdef, #ifndef, #else, #endif, and #undef are implemented.

The functions available with BDS C are reasonably complete with many functions from standard C, such as printf, getc, ungetc, scanf, and fopen. Simple machine-oriented functions that call the CP/M BDOS and the BIOS are available. Peek and poke memory locations, input and output from ports, call assembly language functions, seed and get random numbers, fill a block of memory, move a block of memory, sort a set of elements, execute a CP/M program, and a primitive set of storage allocations functions are also included. Character input and output, with and without formatting is available. Many string and character processing functions are included, as well as file input/output functions. Lastly, a set of plotting functions for DMA video boards such as the Processor Tech VDM-1 are included (I have not tried them).

BDS C compiles in three stages. The first stage is the parser and the second is the code generator. The output of the code generator is a C Relocatable (CRL) file. This CRL file then has to be linked with all of the functions and such to generate the final runable file using CLINK. If no options are specified, CLINK produces a COM file that is executable by CP/M. There are options available in both the compiler and linker that allow programs to be executable at addresses other than 100 Hexadecimal as CP/M requires. This is a great advantage to those people who must write software for other systems or system software for CP/M. The BDS C compiler compiles in one chunk, loading the entire source file into memory at once as opposed to loading the source in segments, working on it a bit, and producing the CRL file. That means that the maximum size of a source file is limited by the size of main memory. This problem can be circumvented by writing programs with a lot of functions and then compiling the functions separately. I have done quite a bit of work with large C programs using many functions, and have had no problem using the compiler in 48K bytes of available memory.

The compiler is simple to use, and compilation and linkage is fast. I have written programs in BDS C and assembler of equal function and found the BDS C compiler to produce executable code in less time than CP/M's assembler. Unfortunately, the assembler produces smaller code than the compiler, since every BDS C program must include approximately 2K of run-time package with every program.

When I first tried to bring up version 1.32 of the compiler, I had no problem. I then received version 1.43 from Leor Zolman, and was unable to link any of the files that I had compiled. I called Leor up and told him of the problem. Since I had to go into Boston anyway, I decided to bring in my non-working version and pick up a working version. I stopped at his place where he gave me a disk that was supposed to be working. I went home to try it out, and when I did, it still didn't work (it proceeded to happily crash my system every time I tried). I called Leor and explained all of the symptoms. He had no idea what was wrong, but offered to make the forty plus mile trip to Worcester (which is horrible in itself) by motorcycle on a very cold day to see if he could figure out what was wrong. To make a long story shorter (I hope), he came out, we tried to figure out what was wrong, and it turned out I had a bad copy of the run-time package.

Other than that one incident, I have been impressed with the operation of the compiler. An interesting note about the BDS implementation is its user following. I have met many people who have been highly impressed by the compiler, and there is also a user group for it. In addition, both the Small C manual and Tom Gibson of Tiny C recommend the BDS C compiler as well. It is a complete and inexpensive implementation of a C compiler

and I would recommend it to anyone intending to do any serious work in the C programming language.

Name: Small C Compiler
Price: $15
Distributed By:
The Code Works
P.O. Box 550
Goleta, CA 93017

The Small C compiler first made its debut in an article published by Ron Cain in the May 1980 issue *Dr. Dobb's Journal*. He opened the article by saying that he had to have a compiler for his home computer. Since Cain did not have CP/M and did not wish to spend the money for the expensive compilers that were on the market, he decided to write his own. He chose to write his 8080 flavored C compiler in C, first using the Tiny c interpreter, then using a full UNIX C compiler. While writing it, he was careful to be sure that all of the code used in writing the compiler would potentially be able to be compiled by that compiler, which would allow it to be modified once installed in the target system. Upon completion of that project, he decided to publish the entire compiler, written in C, in *Dr. Dobbs Journal* along with a discussion of the operation of the compiler. For someone who really needed a compiler, this was great—that is if you had access to a machine with a C compiler on it and if you wanted to type in ten pages of (two columns each) of very, very small print.

At about this time, a company called The Code Works from Goleta, CA entered the scene. They took Ron Cain's Small-C compiler and interfaced it to CP/M. They added to it an unpublished (at that time) run-time system, a few sample programs, and an eleven page instruction manual. They put it all, minus the manual, on a standard CP/M eight inch floppy disk, which included all of the source code for everything, and a working compiled version of the compiler that generated code acceptable to CP/M's assembler. After all that, they started to market it, selling it for an unprecedented $15 plus two dollars postage.

When their press release came to *Microsystems* for publication, I thought something had to be in error. I hadn't seen any compiler software commercially sold for CP/M use at anywhere near that price. When I called them, I was assured that this was in fact the price of the package and not a typographical error.

Once I examined the Small-C package, I was impressed. There was none of the quick, throw-it-together programming and documentation I have found in some of the other software available at that price range. Instead I found a clear, concise manual describing what you could and could not do with this compiler, including a warning that it was not suited for major systems programming tasks. It explained the way its functions were used, and how to interface to assembly language. At the end, the manual gave a specification of what features were available from standard C.

Small C does in fact comply with the "standard" C syntax defined by Kernighan and Ritchie, however its function calls do not comply with standard nor does it support many of the nicer features of C. It has no storage class specifications. The compiler is limited to two byte integer and single byte character types, as well as pointers. In addition, single dimension vector arrays are available. Most unary operators are supported, with the exception of not "!", complement "^ ", and "sizeof". Most binary operators are supported as well, with the exception of connectives and "&&" and or " || ". This means that such statements as:

```
if (a<b && c>d) ++b;
```

cannot be used. The if-else abbreviation "?." is also not available, but is not really a great loss. The concatenated assignment operators "+=", "-=", "*=", "/=", "%=", " > > ", " < < =", "&=", "^ =", and " | =" are not available. This implementation allows very limited structured flow using the "if" and "while" statements. None of the other convenient statements "for," "do while," "switch," "case," "default," and "goto" have been included.

In a small program, many of the above features are not actually necessary, but the lack of ability to logically connect conditions, coupled with a lack of the "for" and "switch-case" operations can make larger programs cumbersome and difficult to program.

As for the I/O functions available, they are rather limited, but usable. Available functions allow single characters and character strings to be read from and written to the user's console. Disk files can be opened and closed, and single byte data can be read in from, or written to any of four open files. Disk I/O is only buffered in 128 byte sectors, however. Another function allows the user to perform CP/M BDOS calls from C.

In addition to the above functions, there is a large amount of generally useful functions inside the compiler source file. It is a shame that these were not pruned from the compiler source and distributed with the other user functions, since they already exist. One such function, called match, compares a specified string with the input stream, and returns true if the two match. This was used in the lexical analysis phase of the compiler, but could just as easily be used in other application programs.

There is a nice feature available that allows inline assembly code to appear in the C source code. Since the compiler generates CP/M 8080 assembler code as its output, inline code can be easily added to the compiler. Any time an "#asm" is encountered in the source, the compiler just takes input and passes it to output with no processing. Once an "#endasm" is encountered, the compiler continues compiling. When I first received the compiler, I had some problems with the assembly code linkages in areas where the manual was unclear, but after a call to The Code Works, I was able to understand it. After my call, they wrote a sample program detailing that linkage, and included it on the distribution disk, so there should be no further problems with other customers.

The Small-C compiler is a one pass compiler (though, technically, the assembler could be considered a second pass). This means that there isn't any checking of defined symbols—everything is just passed on for the assembler to catch. In addition, there is no optimization done on compiled source, either globally or locally. This tends to generate rather inefficient object code, taking more space in memory, and more time to run. In small applications,

this is relatively unimportant, but as the programs increase in size, it can be a real problem.

I found the compiler easy to run, and with no apparent bugs. Unfortunately, compilation is rather sluggish. The run-time package that is included is written in assembler with #asm and #endasm at each end. For a program to assemble, this file must be appended onto the assembler source output of the compiler. The method suggested in the manual is to simply run it through the compiler along with the program being compiled. Doing that is a rather time-consuming process, and I found it much easier to remove the #asm and #endasm statements, and once the compilation was finished, append the run-time software to the end of the compiler output with a text editor. The disadvantage of this is that the entire run-time package is appended, taking up more space in memory.

I was able to compile the compiler source in fourteen minutes on a 4Mhz Z80-A. It took another eight and one half minutes to assemble and 45 seconds to load. After going through all of that, it did run. The source code of the actual compiler was reasonably well written and should be easy to modify. It is also a good example of what a simple, "bare-bones" compiler looks like. In looking at the source, it does not seem that any special parsing or lexical analysis method was used, creating a few additional unnecessary states in code generation.

For anyone looking for a simple compiler to do smaller programming tasks, or for a minimum expense language, the Small-C compiler is a very good choice at $15. For an additional programming tool a few steps above assembler, it is also a good choice. If larger applications are expected, I would not choose this, but go with one of the larger compilers costing from about $145 to $700.

Name: tiny c TWO Compiler
Price: $250
Distributed by:
tiny c associates
P.O. Box 269
Holmdel, NJ 07733

Tiny-C Two Compiler

Tiny-c has been well known for its interpreter of a dialect of the C language, and for its excellent documentation for beginners. After the interpreter was on the market for quite some time, the folks at tiny-c associates felt that although the original tiny-c had quite a following, it was relatively slow for large projects (as an interpreter would be). They decided that a compiler would be very useful.

Now, think about the interpreter compiler combination for a given language. An interpreter is nice and easy to use, and programs can be written, tested, debugged, retested, and so on in the single environment of an interpreter. But interpreters are slow. On the other hand, compilers are faster by at least one order of magnitude, but they are not as simple to work with. You have to enter an editor, type in the program, exit the editor, and attempt to compile. If it won't compile due to syntax errors, you have to reload the editor and edit the program again. When you manage to compile your program, you have to load the linker and debug all of the logic errors. What would offer the best of both worlds would be a good interpreter to make program development easier, and once it works, a compiler to make the program run faster. For those of you who program in Basic, one combination like this is the Microsoft Basic-80 interpreter, which makes development easier. When you want it to cook right along you can use the Microsoft Basic compiler. However, Basic is not the best language for all things, and it would be convenient to have a somewhat structured language do the same thing. The tiny-c 1 interpreter, combined with tiny-c TWO (the compiler), make a strong step in that direction—although the compiler comes with some very powerful functions not supplied in the interpreter.

Tiny-c is a language that is similar to the C programming language. It is important to realize, however, that tiny-c is a different language from C. It has many constructs and data types to those similar in C, but the syntax and operation are slightly different. Where a program in C is written as:

```
main()
{
     some set of operations
}
```

a tiny-c program is written as:

```
main
{
     some set of operations
}
```

Comments in C are written with a beginning /* and end with a */, such as:

```
getch()
/*
     This routine returns a character from the
     console
*/
{
     some operations
}
```

while comments in tiny-c start with /*, and end with the end of the line, such as:

```
getch
/*
/*   This routine returns a character from the
     console
/*
{
     some operations
}
```

In the C language, statements end with a semicolon ';':

```
range=big-little+1;
```

where in tiny-c, the end of a line or a semicolon signifies the end of a statement, although more than one statement may be on a single line is separated by a semicolon:

```
range = big-little+1
last = last*seed

     -or-

range = big-little+1 ; last = last*seed
```

In C, all compound statements are delimited by the curly braces { and } :

```
{
     x=0-1;
     a=b+c;
     b=b*2-e;
     x=b-a;
}
```

In tiny-c they are delimited by square braces [and]:

```
[
     x = x-1
     a = b+c
     b = b*2-a
     c = a-b
]
```

Both allow many levels of nested compound statements.

Functions in C are specified by the function name, a left parenthesis, a list of argument names, and a right parenthesis. This is followed by a type description of each of the arguments:

```
nonprint(c)
char c;
/*
     This function will return true if the character
     is non printable.  It only checks the low order
     seven bits.
*/
{
     return c&128<=32 || c==128;
}
```

It might be called by the statement:

```
if (nonprint(byte))
{
     some statements
}
```

In tiny-c, a function is declared by the name of the function, followed by a type description of each of the arguments.

```
nonprint
char c
/*
/*   This function returns true if the character
/*   is non printable.  It only checks the low order
/*   seven bits.
/*
[
     return c&128<=32
]
```

The function can be called as:

```
if (nonprint(byte))
[
     some statements
]
```

If a function in C returns has no arguments, such as getch, it has to be called like:

```
c=getch();
```

with both parentheses included. In tiny-c, it could be written as:

```
c=getch
```

without the parentheses.

The last major syntactic difference is in the usage of subscripts in arrays. In C, an array is specified by:

```
a=num[index];
```

while in tiny-c:

```
a=num(index)
```

substituting the parenthesis for the square brackets.

For comparison, the following is a small program written in tiny-c and C, taken from the *Tiny-c Newsletter*, Number 1, February, 1981:

tiny-c

```
main
char a(o)
[
     int k
     while (++k < 10)
     [
          pl""
          pn k
          ps ""
          putchar a(k)
     ]
]
```

C

```
main()
char a[o];
{
     int k;
     while (++k < 10)
     {
          pl("");
          pn(k);
          ps("");
          putchar (a[k]);
     }
}
```

The purpose of that newsletter was to explain to users how to convert the tiny-c TWO compiler to a compiler that would compile code closer to standard C. Since the compiler comes with all of the source code, the user only has to make the necessary changes in the code and recompile the compiler. For those who don't wish to make changes themselves, tiny-c associates will be distributing a modified version containing the changes, which they call C-TWO. One of the problems of tiny-c to the experienced C programmer is getting used to and converting old programs to the tiny-c syntax. The C-TWO modification could greatly ease the relearning. For the novice programmer, both to C and to structured languages as a whole, tiny-c is much easier to learn—first with the interpreter, and then with the compiler.

Now that we have looked at the syntactic differences between C and tiny-c, we will look at the functional differences. This is a comparison between the standard C [Kernighan 1978] and tiny-c. Although tiny-c supports many of the features of C, some important items are missing—such as the convenient switch/case statement. In addition, the do/while statement is missing, although the while statement has been implemented. Also missing is the for statement, but this can easily be done in a while loop. A for statement might look like:

```
for( i=1 ; i!=10 ; ++i)
{
     do something
}
```

which can be implemented with a while statement in tiny-c as:

```
i=1
while (i!=10)
[
     do something
     ++i
]
```

It's not as pretty, but it works.

A switch/case statement, on the other hand, is not as easily implemented, nor does it look as nice:

```
switch(num)
{
     case 1:
               statement_1;
               break;          /* exit switch */
     case 2:   statement _2;
               break;
     case 3:   statement _3;
               break;
}
```

This would be done in the form of:

```
if(num==1)
[
   statement_1
]
else
[
     if(num==2)
     [
          statement _2
     [
     else
     [
          if(num==3)
          [
               statment _3
          ]
     ]
]
```

or if not too complex, as:

```
if(num==1) statement_1
else if(num==2) statements_2
else if(num==3) statements_3
```

Tiny-c has the storage classes of static and extern, but does not support auto, register, or typedef. The compiler supports int, long int, and char data types, but does not have floats, doubles, structs, or unions. Tiny-c has a primitive pointer implementation that will address bytes. Normally, in C, when a pointer is declared an int:

```
int *ptr;
```

it points to an integer boundary and, when incremented, it will pass over that integer and point to the next one (usually two bytes later):

```
++ptr;
```

In tiny-c, the pointer will always increment or decrement one byte, regardless of the data type.

Tiny-c also supports many of the unary and binary operators, with the exception of the unary operators type-name, sizeof, and pointer-to (*ptr), in which the latter is implicit when an array is defined. The binary operators "&&" (connective and), " || " (connective or), "=*", "/=", "%=", "+=", "-=", " <<=", " >>=", "&=", "^=", and " |=" are also not implemented. The connective and "&&", and or " || " are the most unfortunate losses in that list, not allowing statements such as:

```
if (a==b && c==d) ++c;
```

Tiny-c comes with a wealth of functions in a function library called TCLIB. Most of the relatively common functions listed in standard C are included, plus a set from the tiny-c interpreter known as training functions. These are a set of functions originally set up for beginners, but are a highly convenient set of functions. For instance:

```
pn x            /* Print number */
```

is much more convenient than:

```
fprintf( so, " %d", x)
```

and tend to be very comfortable to use.

Included in the tiny-c TWO package besides the compiler and run-time system is an operating environment called tiny-shell. The tiny-shell is a mixed blessing to the compiler. It is loosely based on the UNIX shell developed at Bell Telephone Laboratories. In order to do any work with the tiny-c compiler, you have to invoke the shell from CP/M by typing "sh" to CP/M's Console Command Processor. This puts you into a new command processor that executes the tiny-c run-time package and user-written tiny-c programs. There are some strong disadvantages to the shell as well as very strong advantages. Using the tiny-c compiler at the present time, you cannot execute programs directly from CP/M—you first must enter the shell. This means that you cannot create turn-key programs for resale without an agreement with tiny-c associates, although in speaking with Tom Gibson, I discover that the cost of a resale license is reasonable. In addition to not being able to execute tiny-c programs from CP/M, you also cannot execute CP/M programs from the tiny-shell. If you chose to do some work typing in the programs with a text editor like Word Star, you have to exit the tiny-shell first by typing a Control-C character. This has the habit of displaying a rather unfriendly message to the terminal:

```
ERROR # 538 ^C
```

Since the tiny-shell is written in tiny-c, it would have been nice to have the input routine check for a Control-C character and gracefully exit the tiny-shell when encountered. In addition, the operator must become familiar with console input control characters that have slightly different actions in the tiny-shell than in CP/M.

Now for the nice features about the tiny-shell. First, for the programmers who like to define the environment they work in, the tiny-shell (as with everything else in the package) is supplied in source code written in tiny-c. This means that you can modify your environment to your heart's content. In addition, the tiny-shell contains some of the features of I/O redirection found in the UNIX shell. For example, you can write a program that will read in a character from the input device, such as the keyboard, and write it to the output device (the screen). When you run it, it will echo everything typed to the screen.

```
%echo
this is a test
^Z
%
```

Simple, right? Now if you were to use redirection of I/O, and typed:

```
%echo <infile.txt >outfile.txt
%
```

you would copy everything in infile.txt to outfile.txt. The indirection characters "<" for input and ".>" for output provide this powerful and helpful feature. For instance, you can compile a program and get a listing of lines and errors displayed on the terminal. You may not want to wait until it gets to line 187 to find out what is wrong, and dig through all of that output. The solution is to redirect the output to a disk file. You then run the output file through a text editor or searching program to find what you want, without the hassle of all that printout. An additional tiny-shell feature allows the user to type multiple commands on a line, hit return, and have it execute a series of commands without intervention.

CP/M C Compilers

	BDS C	Small-C	Tiny-c TWO	Whitesmiths
Version	1.43	1.1(N)	1	2.0
Price	$145	$15	$250	$630
Language	Assembly	C	Tiny-c	C
Minimum System	32K	24K	32K	60K
Format	most	8"SDSS	8"SDSS	8"SDSS

Addresses:

BDS C:

Lifeboat Associates
1651 Third Avenue
New York, NY 10028

Small-C:

The Code Works
Box 550
Goleta, CA 93017

tiny-c TWO:

tiny c associates
P.O. Box 269
Holmdel, NJ 07733

Whitesmiths C:

Whitesmiths, Ltd.
P.O. Box 1132
Ansonia Station
New York, NY 10023

In addition to the functions built into the tiny shell, later versions of tiny-c TWO come with a set of tiny-c programs that can be compiled into UNIX-like shell functions. These include RM which removes (deletes) a file, MV which moves (renames) a file, LS which lists the directory and gives a status display, CAT which concatenates ASCII files, HD which gives a hex dump of all or part of a file, DIFF which is a file comparator, ED which is a UNIX-like line oriented text editor, and APRINT which types a file to the terminal.

The tiny-c TWO package comes with the source to everything including the runtime package in assembler, the compiler, the linker, the function library, the tiny-shell, and shell functions. In addition, it comes with a one inch thick manual bound in a bright white three ring binder. The manual describes the tiny-c language in a way that is clear to beginning users. It then goes into a description of how all of the functions are called, how to interface to the machine language, how to use the compiler and tiny-shell, examples of tiny-c programs, internal details of the compiler, and installation to other operating systems. The sections of the manual describing the tiny-c language are very clear, but the section on bringing up the compiler on other machines appears to be written in a totally different manner that may not be clear to the novice. I found the manual to be too verbose for an experienced programmer, requiring the entire book to be scanned to learn how the package works. It would have been nice to have a chapter summarizing the syntax, function calls, and compiler operation.

The tiny-c TWO package is useful for the person who has learned the tiny-c language with the interpreter and wants to be able to do production work, or for introducing the beginner to structured programming. It is also very good for the experienced programmer who likes to modify his system to suit his needs because of the sources for the tiny-shell and compiler. For the experienced C programmer who doesn't wish to be tied to a non-system environment, or who wants the power of such things as switch/case or a for loop statement, I would recommend one of the compilers designed for standard C.

Name: Whitesmiths C Compiler
Price: $630
Distributed by:
Whitesmiths, Ltd.
P.O. Box 1132
Anesonia Station
New York, NY 10023

Whitesmiths C Compiler

The Whitesmith C compiler was developed by Whitesmiths, Ltd. of New York City. Their compiler was designed to produce executable programs for the 8080/Z80, LSI-11/PDP-11, VAX-11, M68000. They claim that any program written in C from one machine is portable to any of the other machines, provided that no machine-dependent code is written. This compiler is rather large, and by producing an intermediate code, called A-Natural, can be moved from machine to machine. The A-Natural code (which is assembler-like) can then be compiled/assembled into the executable machine code modules for a given machine.

You could say that the Whitesmiths C compiler for CP/M is the Cadillac of the CP/M C compilers. Like a Cadillac, it is very expensive. At $630, it is the most expensive C compiler available to CP/M. And, as a luxury car eats gasoline, this compiler eats memory. If you don't have 60Kbytes, at minimum, don't even bother to attempt to compile the smallest of C programs. Like the Cadillac, it is cumbersome. A compilation and linkage takes much longer with this compiler than with any others. On the other hand, it tends to generate reasonably fast running code. And also like the Cadillac, it is fancy and has all of the flashy options you could want. The Whitesmiths C compiler complies completely with the "standard," with some extra frills added.

The Whitesmiths package for CP/M comes on two single density floppy disks containing the compiler and machine interface code. In addition to the compilation programs, there is a loader, a library manager, a program to examine relocatable modules, and assembler/compiler to translate the A-Natural code to machine code. There is also a program to translate the A-Natural code into Microsoft Macro Assembler code. In addition to utilities, the Whitesmiths C package comes with a library of 87 high-level functions, seven 8080 C callable subroutines and 55 in-line 8080 subroutines. These final routines are used in the operation of the compiler but can also be used by the systems programmer to do some fancy things like multiplying longs by longs. Finally the package includes twelve functions designed for special calls to the CP/M operating system. At a total of 161 functions available, the programmer has a truly large library to choose from.

Also included in the Whitesmiths C package is a set of two rather thick manuals entitled "C Compiler Users Manual" and "C Compiler Systems Interface Manual for 8080 Users." The two manuals are interesting to read. They vary from lucid to something rivaling IBM's most obtuse literature. Only after the third or fourth reading does it become possible to understand some of the things available and some of the things that are not.

The Whitesmiths C compiler complies completely with the standard in Kernighan and Ritchie, including floating point, longs, and storage classes. In addition, it includes some things not yet in the standard. These include types unsigned (char short, long) and character constants with more than one character. It also provides command redirection using the CP/M console command processor, allowing such things as:

```
A>read program.prn >other.prn
```

which would take all output of program.prn and place it in other.prn. This feature adds about 4Kbytes to the generated object code. The following program is 6K and generates code for command redirection:

```
main()
{
}
```

while this next one is 2K and has no redirection code:

```
_main()
{
}
```

Finding out about the above feature was a true exercise in documentation reading. Every programmer should try it once (and only once).

The Whitesmiths C compiler provides most of the standard formatted I/O and file functions, but generally under different names. For instance, the common function "printf" is called "putfmt" in this implementation. It also contains functions that operate on arguments of the command line, many string, numerical and data conversion functions.

Although this C compiler produces reasonably fast code, compilation and linking requires a great deal of patience. It compiles in five steps, generating intermediate files, using the CP/M submit facility. The first step is the preprocessor (pp) followed by the parser (p1) and then the 8080 code generator (p2). Once it has generated the A-Natural source code, it must go through the A-Natural assembler/compiler. Finally it must go through a very long linkage process to bring together all the functions and produce executable code.

This compiler has many tradeoffs, but in certain cases it is very valuable to the system developer. On the negative side are documentation that is not entirely clear, the very long time for edit, compile, test, and edit runs, and its requirement for a huge amount of memory. I had to buy a special chip set for my disk controller just to test this compiler. Also the licensing and ordering processes are very comfortable. After signing a very extensive two page license agreement, I received the compiler. From what I understand from the license agreements, even with the payment of over six hundred dollars, Whitesmiths, Ltd. still owns the thing and can require it to be returned any time in the next fifteen years.

On the positive side, the compiler generates fast code which contains the complete C syntax and is thus portable to other machines. This allows a program, such as a compiler, to be developed on one machine (like the PDP-11) and moved either up to the VAX or down to the 8080/Z80 machines (provided you buy three versions of the compiler, of course).

The Whitesmiths C compiler is definitely not the compiler for a casual user. It is however, quite useful and possibly necessary to the person who needs to compile large, production-type programs using the full C syntax.

An Introduction to the C Programming Language — Part 2

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.

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 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.

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.

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

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.

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.

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

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 reviewing 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*k07+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;
                }

/*

                Performance Evaluation Program #4

                        Pointer Operations

                          (BDS C)

*/

main()
{
        char arry[128], *xptr;
        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");
}
                else
                {
                        j = 2500;
                }

        }
        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 n;
}


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

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");
}


/*
                Performance 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"
}


/*              Performance 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 x
{
    return x;
}
```

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*807+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");
}


/*
                Performance 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!=600; ++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!=600; ++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;
}
```

Chapter V
Utilities/Enhancements

An Improved CP/M BIOS for the Tarbell Disk Controller

Martin Nichols

The following program is an improved BIOS (Basic Input/Output System) for CP/M when used with a Tarbell single density disk controller. It offers several improvements over the original BIOS provided with the Tarbell Disk controller. For example, it will support 4 disk drives, compared to 2 previously and provides faster disk I/O operations. Further, it includes a driver for VDM-type CRT displays.

The source code is also extremely well documented allowing for easy modification by the user. Those users wishing the source code on an 8" disk can obtain it by writing to the author. The charge for the disk is $10.

Martin Nichols, 100 Guy St., Dover, NJ 07801.

```
;*****************************************************
;
;BASIC INPUT/OUTPUT OPERATING SYSTEM
;FOR USE WITH THE TARBELL CONTROLLER
;
;
;THIS MODULE CONTAINS ALL THE INPUT/OUTPUT
;ROUTINES FOR THE CP/M SYSTEM, INCLUDING
;THE DISK ROUTINES.
;
;THIS VERSION SUPPORTS THE FOUR DRIVE SYSTEM
;OF CP/M 1.4. IT CAN ALSO BE USED WITH 1.3.
;THE ONLY CHANGE NEEDED IF YOU ARE USING 1.3 IS
;TO CHANGE THE BDOS EQUATE FROM "CBASE+3106H"
;TO "CBASE+3206H".
;
;****** THIS BIOS REQUIRES 1K MORE THAN THE 512 BYTES
;ALLOWED BY DIGITAL RESEARCH.  THEREFORE, IF YOU WANT
;TO RUN THIS BIOS YOU WILL HAVE TO GENERATE A COM THAT
;IS 1K LESS THAN THE AMOUNT OF MEMORY THAT YOU WANT IT
;TO RUN IN (I.E. 23 TO FIT IN 24K, 31 FOR 32K, ETC).
;MSIZE SHOULD AGREE TO THE NUMBER OF K YOU SPECIFIED
;TO MOVCPM OR RELOC (23 FOR 24K SYSTEM).  USE THE
;BIOS FORMULA GIVEN IN DIGITAL RESEARCH MANUAL.
;
;IF YOU WANT VDM AS STD OUTPUT-"PTVDM" SHOULD BE TRUE.
;IF YOU WANT STANDARD OUTPUT ROUTINE-"PTVDM" SHOULD BE FALSE.
;
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;
; "MYSYS" MUST BE FALSE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
;
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;
;
;
;THE DISK DRIVER ROUTINES IN THIS BIOS ARE DEPENDENT
;ON EACH OTHER FOR CERTAIN INFORMATION AND TIMINGS.
;MODIFICATIONS TO THE DISK DRIVER ROUTINES SHOULD
;ONLY BE DONE AFTER CAREFUL STUDY.
;
;THIS VERSION CONTAINS SUPPORT FOR FAST READING AND
;WRITING ON A NON-INTERLEAVED BASIS. THAT IS, IT CAN
;READ OR WRITE A COMPLETE TRACK IN ONE REVOLUTION OF
;THE DISK IF THE READS OR WRITES ARE ISSUED IN
;SEQUENTIAL ORDER.
;
;IF YOU HAVE ONLY ONE DISK CHANGE "ONEDSK" TO TRUE.
;THE I/O SYSTEM WILL THEN SIMULATE A MULTIPLE DRIVE SYSTEM.
;
;ERROR MESSAGES ARE PRINTED IN THE FOLLOWING FORMAT:
;
;XAAA TNN SNN X
;
;       WHERE X = (R)READ,(W)WRITE
;       AND AAA = NRY - NOT READY - BIT 7
;                 WPR - WRITE PROTECT - BIT 6
;                 WFT - WRITE FAULT - BIT 5
;                 RNF - RECORD NOT FOUND - BIT 4
;                 CRC - CRC ERROR - BIT 3
;                 LDA - LOST DATA - BIT 2
;                 DRQ - DATA REQUEST - BIT 1
;                 BSY - BUSY - BIT 0
;           TNN = TRACK WHICH CONTAINS ERROR (DECIMAL)
;           SNN = SECTOR WHICH CONTAINS ERROR (DECIMAL)
;             X = DRIVE ON WHICH ERROR OCCURED
;       THE BIT POSITIONS ARE GIVEN SO THAT THE 1771 MANUAL
;       COULD BE CONSULTED FOR FURTHER EXPLANATION.
;
;---WRITTEN BY M. D. NICHOLS---
;
;*****************************************************

;
;FREQUENTLY CHANGED PARAMETERS
;
TRUE    EQU     0FFFFH          ;USED IN IF STATEMENTS
FALSE   EQU     NOT TRUE        ;USED IN IF STATEMENTS
PTVDM   EQU     TRUE            ;VDM FOR CONOT?
ONEDSK  EQU     FALSE           ;USE IF YOU ONLY HAVE 1 DRIVE
TEST    EQU     TRUE            ;TESTING BIOS
MSIZE   EQU     47              ;MEMORY SIZE
MODIF   EQU     'G'             ;MODIF LEVEL OF DRIVER
SUBMOD  EQU     '4'             ;SUB MODIFICATION LEVEL
MYSYS   EQU     TRUE            ;SPECIAL STUFF FOR MY SYSTEM
;
DBASE   EQU     0F8H            ;BASE FOR CONTROLLER PORTS
DCOM    EQU     DBASE           ;DISK COMMAND PORT
DSTAT   EQU     DBASE           ;DISK STATUS PORT
DTRK    EQU     DBASE+1         ;DISK TRACK PORT
DSECT   EQU     DBASE+2         ;DISK SECTOR PORT
DDATA   EQU     DBASE+3         ;DISK DATA PORT
DWAIT   EQU     DBASE+4         ;DISK WAIT PORT
DCONT   EQU     DBASE+4         ;DISK CONTROL PORT
        ORG     MSIZE*1024-512  ;THIS SETS ADDR OF CBIOS
CBASE   EQU     (MSIZE-35)*1024 ;TEMPORARY CALCULATION
CCP     EQU     CBASE+1600H     ;START OF CP/M SYSTEM
BDOS    EQU     CBASE+1806H     ;START OF BDOS FOR 1.4
CPML    EQU     $-CCP           ;LENGTH OF CP/M W/O CBIOS
NSECTS  EQU     CPML/128+26     ;# SECT CP/M EXCL. CBIOS
RTCNT   EQU     10              ;RETRY COUNT ON ERRORS
;
        JMP     BOOT            ;PRIM COLD START LOADER
XWBOOT  JMP     WBOOT           ;ERUM WARM START
XCONST  JMP     CONST           ;CHECK CONSOLE STAT
XCONIN  JMP     CONIN           ;GET CONSOLE CHARACTER
XCONOT  JMP     CONOT           ;WRITE CONSOLE CHARACTER
XLIST   JMP     LIST            ;WRITE CHAR ON LIST DEV
XPUNCH  JMP     PUNCH           ;PUNCH CHAR ON PAPER TAPE
XREAD   JMP     READER          ;READ CHAR FROM PAPER TAPE
XHOME   JMP     HOME            ;RESTORE HEAD TO TRK 0
```

```
XDSKSEL JMP     DSKSEL          ;SELECT DISK DRIVE
XSETTRK JMP     SETTRK          ;SET TRACK NUMBER
XSETSEC JMP     SETSEC          ;SET SECTOR NUMBER
XSETDMA JMP     SETDMA          ;SET DMA ADDRESS
XREAD   JMP     READ            ;READ SELECTED SECTOR
XWRITE  JMP     WRITE           ;WRITE SELCECTED SECTOR
;
;ENTER HERE FROM COLD START LOADER
;GIVE SIGN ON MESSAGE
;
BOOT:
        LXI     SP,80H          ;SET STACK POINTER
        XRA     A               ;CLEAR A
        STA     4               ;FORCE DISK A
;
        IF      MYSYS OR PTVDM
        OUT     0C8H            ;RESET VDM
        MVI     C,0CH           ;SET UP FF TO CLEAR SCREEN
        CALL    XCONOT          ;SEND IT OUT
        ENDIF
;
        LXI     H,SMSG          ;PRINT OPENING MESSAGE
        CALL    PMSG            ;DO PRINTOUT
        JMP     GOCPM           ;NOW DO HOUSEKEEPING
;
;WARM BOOT ROUTINE--ACTIVATED WHEN YOU CONTROL C
;
WBOOT:
        LXI     SP,80H          ;SET STACK POINTER
WBOOT1:
        MVI     C,0             ;USED BY DSKSEL AND SETTRK
        CALL    XDSKSEL         ;SELECT DISK 0
        MVI     D,NSECTS        ;# OF SECTORS IN D
        MVI     B,2             ;START WITH SECTOR 2
        LXI     H,CCP           ;GET STARTING ADDR
WBOOT2:
        CALL    SETTRK          ;SELECT TRACK (C)
        PUSH    B               ;SAVE BC
        MOV     C,B             ;PUT SECTOR IN C
        CALL    XSETSEC         ;SET IT UP
        MOV     B,H             ;GET READY FOR SET DMA
        MOV     C,L             ;
        CALL    XSETDMA         ;SET IT UP
        POP     B               ;RESTORE BC
        CALL    XREAD           ;READ RECORD
        JNZ     WBOOTX          ;ERROR ON READ
        DCR     D               ;# OF SECTORS TO READ - 1
        JZ      GOCPM           ;DONE--GO TO CPM
        INR     B               ;POINT TO NEXT SECTOR
        MOV     A,B             ;MOVE IT TO A
        CPI     27              ;END OF TRACK
        JC      WBOOT2          ;NO--CONTINUE READING
        INR     C               ;TRACK NOW EQUAL 1
        MOV     B,C             ;SET SECTOR BACK TO 1
        JMP     WBOOT2          ;READ NEXT TRACK
WBOOTX:
        LXI     H,BTMSG         ;GET ADDR OF ERROR MSG
        CALL    PMSG            ;PRINT IT
        CALL    XCONIN          ;WAIT FOR KB ENTRY
        JMP     WBOOT1          ;DO WARM BOOT AGAIN
;
;THIS ROUTINE IS THE EXIT TO CP/M SYSTEM
;
GOCPM:
        MVI     A,0C3H          ;PUT JUMP TO WARM BOOT
        STA     0               ;AT ADDR ZERO
        LXI     H,WBOOT         ;GET ADDR OF WARM BOOT ENTRY
        SHLD    1               ;FINISH THE JUMP INSTRUCTION
        STA     5               ;PUT JUMP TO BDOS AT 5
        LXI     H,BDOS          ;GET ADDR OF BDOS ENTRY
        SHLD    6               ;FINISH JUMP INSTRUCTION
        STA     38H             ;SET UP INTERRUPT TRAP
        LXI     H,TRAP          ;
        SHLD    39H             ;AND IT
        LXI     H,80H           ;SET DEFAULT DMA ADDR
        SHLD    DMAADD          ;AND STORE IT
        LDA     4               ;GET PREV DRIVE #
        MOV     C,A             ;SHOULD BE IN C
;
        IF      TEST            ;
        XRA     A               ;CLEAR A
        OUT     0FFH            ;SHOW IN TIM FP LIGHTS
        ENDIF
;
        JMP     CCP             ;GO TO CP/M
;
;THIS ROUTINE IS A NULL INTERRUPT TRAP
;FOR NOW, JUST RE-ENABLE INTERRUPTS
;
TRAP:
        DB      0               ;LEAVE ROOM FOR JUMP OR CALL
        DB      0               ;
        DB      0               ;
        EI                      ;TURN INTERRUPTS BACK ON
        RET                     ;GO BACK FROM WHENCE YOU CAME
;
;SELECT DISK GIVEN BY REG C
;
DSKSEL:
        MOV     A,C             ;PUT NEW DISK IN A
        STA     NXTDSK          ;SAVE IT FOR REAL RTN
        RET                     ;
;
;SELECT DISK STORED IN NXTDSK
;
SELDSK:
        PUSH    H               ;
        PUSH    D               ;SAVE REGS
        PUSH    B               ;
        LDA     NXTDSK          ;GET NEW DISK #
        MOV     C,A             ;
        ANI     3               ;LOOK AT 2 LSB'S
        LXI     H,DISKNO        ;GET ADDR OF DISKNO
        CMP     M               ;NEW=OLD
        JZ      SELXIT          ;IF SO,RETURN
;
        IF      ONEDSK
;
        PUSH    B               ;SAVE REGS
        PUSH    A               ;SAVE REG$
        LXI     H,MNTMSG        ;PRINT 'MOUNT' MSG
        CALL    PMSG            ;TELL THEM
        POP     A               ;RETRIEVE REG A
        STA     DISKNO          ;STORE IT FOR LATER
        ADI     'A'             ;MAKE IT ASCII LETTER
        MOV     C,A             ;PUT IN C FOR CONOT
        CALL    XCONOT          ;PRINT IT
        CALL    XCONIN          ;WAIT FOR GO AHEAD
        POP     B               ;RESTORE BC
        XRA     A               ;CLEAR A
        JMP     SELXIT          ;EXIT SELDSK
;
        ENDIF
;
        MOV     E,M             ;PUT OLD DISK # IN DE
        MVI     D,0             ;CLEAR D FOR DAD
        LXI     H,TRKTB         ;GET ADDR OF TRACK TABLE
        DAD     D               ;ADD DISK # TO ADDR
        IN      DTRK            ;GET TRACK FROM OLD DRIVE
        MOV     M,A             ;STORE IT IN TRACK TABLE
        MOV     E,C             ;GET NEW DRIVE #
        LXI     H,TRKTB         ;GET ADDR OF TRK TABLE
        DAD     D               ;GET HEAD LOC ON NEW DRIVE
        MOV     A,M             ;AND PUT IN REG A
        STA     HOLD            ;SAVE FOR SEEK ROUTINE
        OUT     DTRK            ;ADJUST 1771
        MOV     A,C             ;GET DISK #
        STA     DISKNO          ;STORE IT FOR LATER USE
        CMA                     ;INVERT BIT FOR LATCH
        ADD     A               ;PUT BITS 0-1 AT 4-5
        ADD     A               ;
        ADD     A               ;
        ADD     A               ;
        ORI     2               ;MAKE LATCH COMMAND
        OUT     DCONT           ;SET LATCH WITH CODE
SELDEL:
        IN      DSTAT           ;UNLOAD HEAD BECAUSE 1771 DOES
        ANI     20H             ;NOT RECOGNIZE DRIVE SWITCH
        JNZ     SELDEL          ;
        XRA     A               ;SAY EVERYTHING OK
SELXIT:
        POP     B               ;RESTORE REGS
        POP     D               ;RESTORE REGS
        POP     H               ;RESTORE REGS
        RET                     ;
;
;SET THE TRACK GIVEN IN REG C
;
SETTRK:
        MOV     A,C             ;TRK WAS IN REG C
        STA     TRK             ;PUT IT WHERE IT CAN BE FOUND
        RET                     ;
;
;SET DISK SECTOR NUMBER
;
SETSEC:
        MOV     A,C             ;GET SECTOR NUMBER
        STA     SECT            ;PUT AT SECTOR # ADDR
        RET                     ;
;
;SET DISK DMA ADDRESS
;
SETDMA:
        MOV     H,B             ;MOV BC TO HL
        MOV     L,C             ;
        SHLD    DMAADD          ;SAVE IT
        RET                     ;
;
;READ A SECTOR AT SECT, SEEK THE NEEDED TRACK
;USE STARTING ADDRESS AT DMAADD
;
READ:
        CALL    SEEK            ;MOVE HEAD TO TRACK
        MVI     A,RTCNT         ;GET RETRY CNT
RRETRY:
        STA     ERCNT           ;SAVE ERROR CNT
        LDA     SECT            ;GET SECTOR NUMBER
        LHLD    DMAADD          ;GET STARTING ADDRESS
READ1:
        OUT     DSECT           ;SET SECTOR IN 1771
        CALL    RDYCK           ;SEE IF DRIVE READY
        DI                      ;DO NOT ALLOW INTERRUPTS
        IN      DSTAT           ;GET STATUS
        ANI     20H             ;CHECK IF HEAD LOADED
;
        MVI     A,88H           ;SETUP READ M/D HEAD LOAD
        JNZ     READE           ;IF LOADED - THEN DO IT
        ORI     4               ;ELSE FORCE HEAD LOAD
READE:
        OUT     DCOM            ;SEND COMMAND TO 1771
RLOOP:
        CALL    FREAD           ;READ A SECTOR
RDDONE:
        EI                      ;ALLOW INTERRUPTS AGAIN
        IN      DSTAT           ;READ DISK STATUS
        ANI     9DH             ;LOOK AT ERROR BITS
        RZ                      ;RETURN IF NONE
CHECK:
        STA     ERRS            ;SAVE ERROR BYTE
;
        IF      TEST            ;
        CMA                     ;FP INVERTED LOGIC
        OUT     0FFH            ;IN THE LIMELIGHT
        ENDIF                   ;
```

```
;
        CALL    ERCHK           ;CHECK FOR SEEK ERROR
        LDA     ERCNT           ;GET ERROR CNT
        DCR     A               ;DECREMENT COUNT
        JNZ     RRETRY          ;TRY TO READ AGAIN
        MVI     A,'R'           ;SHOW READ ERROR
        JMP     ERRMSG          ;TELL SOMEONE
FREAD:
        IN      DWAIT           ;WAIT FOR DRQ OR INTRQ
        ORA     A               ;SET FLAGS
        RP                      ;DONE IF INTRQ
        IN      DDATA           ;READ A BYTE FROM DISK
        MOV     M,A             ;PUT BYTE IN MEMORY
        INX     H               ;INCR MEMORY POINTER
        JMP     FREAD           ;KEEP READING
;
;WRITE THE SECTOR AT SECT--LOAD HEAD FIRST
;USE STARTING ADDRESS AT DMAADD
;
WRITE:
        CALL    SEEK            ;GET ON RIGHT TRACK
        MVI     A,RTCNT         ;GET RETRY COUNT
WRETRY:
        STA     ERCNT           ;SAVE ERROR CNT
        LDA     SECT            ;GET SECTOR #
        LHLD    DMAADD          ;GET STARTING ADDR
WRITE1:
        OUT     DSECT           ;SET SECT INTO 1771
        CALL    RDYCK           ;SEE IF DRIVE READY
        DI                      ;DO NOT ALLOW INTERRUPTS
        IN      DSTAT           ;GET DISK STATUS
        ANI     20H             ;CHECK FOR HEAD LOADED
        MVI     A,0A8H          ;SETUP WRITE W/O HEAD LOAD
        JNZ     WRITEN          ;IF LOADED THEN DO IT
        ORI     4               ;FORCE WRITE WITH HEAD LOAD
WRITEN:
        OUT     DCOM            ;ISSUE COMMAND
WLOOP:
        CALL    FWRITE          ;WRITE A SECTOR
WDONE:
        EI                      ;ALLOW INTERRUPTS AGAIN
        IN      DSTAT           ;READ DISK STATUS
        ANI     0FDH            ;MASK NON-ERROR BITS
        RZ                      ;RETURN IF NO ERRORS
        STA     ERRS            ;SAVE ERROR FLAGS
;
        IF      TEST
        CMA                     ;INVERT THEM
        OUT     0FEH            ;PUT THEM ON FP LEDS
        ENDIF                   ;
;
        CALL    ERCHK           ;CHK/CORRECT SEEK ERR
        LDA     ERCNT           ;GET ERROR CNT
        DCR     A               ;DECREMENT COUNT
        JNZ     WRETRY          ;TRY WRITE AGAIN
        MVI     A,'W'           ;SHOW WRITE ERROR
        JMP     ERRMSG          ;DO ERROR MESSAGES
FWRITE:
        IN      DWAIT           ;WAIT FOR READY
        ORA     A               ;SET FLAGS
        RP                      ;GET OUT WHEN DONE
        MOV     A,M             ;GET BYTE FROM MEMORY
        OUT     DDATA           ;WRITE ON DISK
        INX     H               ;POINT TO NEXT BYTE
        JMP     FWRITE          ;KEEP WRITING
;
;
;READ OR WRITE ERROR DETECTED--HANDLE NO REC FOUND
;CONDITION ELSE NORMAL RETRY LOOP
;
ERCHK:
        LDA     ERRS            ;GET ERROR BYTE
        ANI     10H             ;MASK FOR NRF
        RZ                      ;NOT NRF FAULT
;
;CHECK TO SEE IF ON CORRECT TRACK
;CORRECT IF NECESSARY
;
CHKSK:
        MVI     A,0C4H          ;SET UP READ ADDR
        OUT     DCOM            ;COMMAND TO 1771
        DI                      ;DO NOT ALLOW INTERRUPTS
        IN      DWAIT           ;WAIT FOR 1ST DRQ (TRK)
        IN      DDATA           ;READ THE TRACK ADDR
        EI                      ;ALLOW INTERRUPTS AGAIN
        PUSH    PSW             ;SAVE TRACK
        CALL    SWAIT           ;WAIT FOR OPERATION TO FINISH
        POP     PSW             ;GET TRACK BACK
        STA     HOLD            ;USE IT TO SET UP SEEK RTN
        OUT     DTRY            ;UPDATE TRACK REGISTER
;
;TRACK DESIRED HAS ALREADY BEEN STORED BY SETTRK
;
SEEK:
        CALL    SELOK           ;WILL DO NOTHING IF FROM CHKSK
        LDA     TRK             ;GET WHERE WE ARE GOING TO
        PUSH    B               ;SAVE BC
        MOV     B,A             ;SAVE TRK IN B
        LDA     HOLD            ;GET PREV TRACK
        CMP     B               ;COMPARE TO CURRENT
        JZ      FORINT          ;JUMP SO FORCE INTERRUPT
        MOV     A,B             ;PUT CURRENT IN A
        STA     HOLD            ;SAVE FOR NEXT TIME
        POP     B               ;RESTORE BC
        CPI     0               ;SEE IF TRK = 0
;
        JZ      HOMRTN          ;DO HOME ROUTINE INSTEAD
        OUT     DDATA           ;GIVE DESIRED TRK TO 1771
        CALL    SWAIT           ;WAIT TILL NOT BUSY
        MVI     A,1AH           ;SEEK-I&V&S-
        CALL    SCMND           ;ISSUE COMMAND
        CALL    SLOOP           ;GIVE TIME FOR HEAD TO SETTLE
        RET                     ;
;
SCMND:
        OUT     DCOM            ;ISSUE COMMAND
SWAIT:
        CALL    ZIP             ;WE NEED AT LEAST 12 US
        CALL    ZIP             ;AGAIN, FOR 4 MHZ
SBUSY:
        IN      DSTAT           ;WAIT FOR NOT BUSY, IE INTRQ
        RRC                     ;SET CARRY IF BUSY
        JC      SBUSY           ;STILL BUSY
ZIP:
        RET                     ;INTRQ RESET BY READING STATUS
;
FORINT:
        MVI     A,0D0H          ;CLEAR ANY PENDING COMMAND
        OUT     DCOM            ;AND FORCE TYPE I STATUS
        POP     B               ;CLEAN UP
        RET                     ;GO BACK
;
SLOOP:
        PUSH    H               ;GIVE TIME FOR HEAD TO SETTLE
        LXI     H,3*256         ;NEED ABOUT 30 MS
SLOOP1:
        DCR     L               ;
        JNZ     SLOOP1          ;
        DCR     H               ;
        JNZ     SLOOP1          ;
        POP     H               ;
        RET                     ;
;HOME ROUTINE-SET TRK TO ZERO AND SEEK
;WILL PICK UP DURING READ OR WRITE
;
HOME:
;
        XRA     A               ;CLEAR A
        STA     TRK             ;PUT WHERE IT CAN BE FOUND
        RET                     ;
;
;THIS ROUTINE ONLY CALLED FROM SEEK
;TO PERFORM ACTUAL HOME
;
HOMRTN:
        MVI     A,0D0H          ;RESET ANY PENDING COMMAND
        OUT     DCOM            ;ISSUE COMMAND
        CALL    SWAIT           ;WAIT FOR NOT BUSY
        MVI     A,0AH           ;10 MS SEEK RATE--HOME
        CALL    SCMND           ;ISSUE COMMAND-WAIT FOR INTRQ
;
        CALL    SLOOP           ;FOR HEAD SETTLING
        RET                     ;GO BACK
;
;CHECK FOR DRIVE READY-IF NOT TELL OPERATOR
;AND WAIT FOR CR
;
RDYCK:
        IN      DSTAT           ;GET DISK STATUS
        ANI     80H             ;MASK FOR READY
        RZ                      ;OK WE ARE READY
        PUSH    H               ;SAVE REGS
        PUSH    B               ;SAVE REGS
RDY1:
        LXI     H,NRDYMS        ;POINT TO MESSAGE
        CALL    PMSG            ;PRINT IT OUT
        LDA     DISKNO          ;GET CURRENT DRIVE
        ADI     'A'             ;CHANGE TO ASCII
        MOV     C,A             ;SET UP TO PRINT
        CALL    XCONOT          ;PRINT IT
        CALL    XCONIN          ;GET KEYBD CHAR
        LXI     H,CRLF          ;SET UP CR AND LF
        CALL    PMSG            ;PRINT IT
        POP     B               ;RESTORE REGS
        POP     H               ;RESTORE REGS
        RET                     ;
;
;
;PRINT MESSAGE ROUTINE
;
PMSG:
        MOV     A,M             ;GET BYTE TO PRINT
        ORA     A               ;SEE IF BINARY ZERO
        RZ                      ;YES, WE ARE DONE
        MOV     C,A             ;PASS IT IN C
        CALL    XCONOT          ;PRINT A CHAR
        INX     H               ;POINT TO NEXT BYTE
        JMP     PMSG            ;STAY IN LOOP TILL DONE
;
;ERROR MESSAGE ROUTINES
;
ERRMSG:
        PUSH    H               ;SAVE HL
        PUSH    B               ;SAVE BC
        MOV     B,A             ;SAVE TYPE INDICATOR
        LXI     H,CRLF          ;DO CRLF
        CALL    PMSG            ;PRINT IT
        MOV     C,B             ;GET TYPE INDIC
        CALL    XCONOT          ;PRINT IT
        LDA     ERRS            ;GET ERROR BYTE
        LXI     H,TYPERR-1      ;POINT TO ERROR TABLE
LOCERR:
        INX     H               ;POINT TO NEXT ENTRY
        INX     H               ;IN ERROR TYPE TABLE
        INX     H               ;
        RLC                     ;SHIFT BIT INTO CARRY
        JNC     LOCERR          ;NOT IN ERROR-KEEP LOOKING
        MVI     B,3             ;SET B = 3
TYPRNT:
        MOV     C,M             ;GET FIRST LETTER
        CALL    XCONOT          ;PRINT IT
        INX     H               ;POIN T TO NEXT LETTER
        DCR     B               ;DECREMENT COUNTER
        JNZ     TYPRNT          ;SEE IF MORE LETTERS
        CALL    BLK             ;PRINT A BLANK
        MVI     C,'T'           ;PRINT A T FOR TRACK
```

```
        LDA     TRK             ;GET TRACK WE ARE ON
        CALL    ERRPRT          ;SHOW THEM LOCATION
        MVI     C,'S'           ;PRINT AN S FOR SECTOR
        LDA     SECT            ;GET SECT WE ARE ON
        CALL    ERRPRT          ;AND SHOW IT
        LDA     DTSKNO          ;GET CURRENT DRIVE
        ADI     'A'             ;MAKE IT ASCII
        MOV     C,A             ;SET UP FOR PRINT
        CALL    XCONOT          ;PRINT IT
        POP     B               ;RESTORE B
        POP     H               ;RESTORE HL
        MVI     A,1             ;SIGNAL ERROR
        ORA     A               ;SET FLAGS
        RET                     ;GO HOME
ERRPRT:
        PUSH    PSW             ;SAVE NUMBER
        CALL    XCONOT          ;PRINT LETTER IN C
        POP     PSW             ;GET NUMBER BACK
DECPRT:
        MVI     C,'0'-1         ;SET UP C FOR 10'S DIGIT
DECPR1:
        INR     C               ;EXTRACT MS DIGIT
        SUI     10              ;BY REPETITIVE SUBTRACTION
        JP      DECPR1          ;
        ADI     10+'0'          ;RESTORE TO POSITIVE
        MOV     B,A             ;SAVE LS DIGIT
        CALL    XCONOT          ;LIST TENS DIGIT
        MOV     C,B             ;PUT UNITS DIGIT IN C
        CALL    XCONOT          ;PRINT A BLANK
BLK:
        MVI     C,' '           ;PRINT A BLANK
        JMP     XCONOT          ;PRINT IT AND RETURN
;
;CBIOS MESSAGES
;
SMSG:   DB      0DH,0AH,'MDM CP/M R1.V4 /'
        DB      MODIF,SUBMOD,'/',0DH,0AH,0
BTMSG:  DB      0DH,0AH,'BOOT FAILURE',0
CRLF:   DB      0DH,0AH,0
MNTMSG: DB      0DH,0AH,'MOUNT ',0
NRDYMS  DB      0DH,0AH,'NOT READY-DRIVE ',0
TYPERR: DB      'NRYWPRMFTRNFCRCLDADRQRSY'
;
;CHECK CONSOLE INPUT STATUS
;NOT READY (A)=0, READY (A)=FF

;
CONST:
;
        IF      MYSYS           ;
        IN      2               ;READ CONSOLE STATUS
        ANI     80H             ;LOOK AT BIT 7
        ENDIF                   ;
;
        IF      NOT MYSYS       ;
        IN      0               ;
        ANI     1               ;LOOK AT BIT 0
        ENDIF                   ;
;
        MVI     A,0             ;SET A=0
        RNZ                     ;RETURN WITH NOT READY
        CMA                     ;READY SO A=FF
        RET                     ;
;
;READ A CHARACTER FROM CONSOLE
;
CONIN:
;
        IF      MYSYS           ;
        IN      2               ;READ CONSOLE STAT
        ANI     80H             ;LOOK AT BIT 7
        JNZ     CONIN           ;KEEP WAITING
        IN      0               ;GET DATA BYTE
        ANI     7FH             ;TURN PARITY OFF
        RET                     ;
        ENDIF                   ;
;
        IF      NOT MYSYS       ;
        IN      0               ;READ CONSOLE STAT
        ANI     1               ;LOOK AT BIT 0
        JNZ     CONIN           ;NOT RDY-KEEP WAITING
        IN      1               ;READ DATA BYTE
        ANI     7FH             ;TURN OFF PARITY
        RET                     ;
        ENDIF                   ;
;
;WRITE CHARACTER IN (C) TO CONSOLE DEVICE
;
CONOT:
;
        IF      NOT PTVDM
        IN      0               ;GET STATUS
        ANI     80H             ;LOOK AT BIT 7
        JNZ     CONOT           ;NOT READY-KEEP WAITING
        MOV     A,C             ;CHAR IN C TO A
        OUT     1               ;PRINT IT
        RET                     ;
        ENDIF                   ;
;
        IF      PTVDM           ;
;
;
VDBASE  EQU     0C000H          ;SCREEN MEMORY AREA
VDBAS2  EQU     VDBASE/256      ;MSB OF VDBASE
;
;
        PUSH    H               ;SAVE HL
        LHLD    VDMP            ;GET CURRENT CURSOR LOC
        CALL    VDM             ;DO OUTPUT
        SHLD    VDMP            ;SAVE CURSOR LOCATION
        POP     H               ;RESTORE HL
        RET                     ;

;
VDM:
        MOV     A,C             ;PUT CHAR IN A
        ANI     7FH             ;TURN OFF PARITY
        CPI     7FH             ;CHECK FOR RUBOUT
        JZ      EXIT            ;DO NOTHING IF RUBOUT
        MOV     A,M             ;GET CURSOR
        ANI     7FH             ;TURN IT OFF
        MOV     M,A             ;PUT BACK ON SCREEN
        MOV     A,C             ;GET CHAR IN A
        CPI     ' '             ;CHECK FOR CONTROL CHAR
        JC      CTLCHR          ;IF YES, DO SOMETHING SPECIAL
        MOV     M,A             ;PUT ON SCREEN
        INX     H               ;POINT TO NEXT LOCATION
        MOV     A,H             ;GET SCREEN ADDR MSB
        CPI     VDBAS2+4        ;CHECK FOR END OF SCREEN
        CZ      SCRL            ;IF YES, THEN SCROLL
        JMP     EXIT            ;GET OUT OF HERE
SCRL:
        LXI     H,VDBASE        ;POINT TO START OF SCREEN
        PUSH    B               ;SAVE BC
        LXI     B,VDBASE+64     ;PUT START + 1 LINE IN B
SCRL1:
        LDAX    B               ;GET BYTE FROM SCREEN
        MOV     M,A             ;AND PUT IN NEW POSITION
        INX     B               ;INCREMENT POINTERS
        INX     H               ;
        MOV     A,B             ;GET ADDR MSB
        CPI     VDBAS2+4        ;CHECK FOR END OF SCREEN
        JNZ     SCRL1           ;IF NOT, THEN KEEP DOING
        POP     B               ;RESTORE BC
        PUSH    H               ;SAVE THIS ADDR
SCRL2:
        MVI     M,' '           ;PUT BLANK ON LAST LINE
        INX     H               ;POINT TO NEXT LOC
        MOV     A,L             ;GET ADDR LSB
        ANI     3FH             ;CHECK END-OF LINE
        JNZ     SCRL2           ;NOT YET
        POP     H               ;POINT TO BEGIN OF LINE
        RET                     ;WE ARE DONE
CTLCHR:
        CPI     08H             ;IS IT BS
        JZ      DELETE          ;YES
        CPI     0DH             ;IS IT CR
        JZ      CRTN            ;YES
        CPI     0AH             ;IS IT LF
        JZ      CLF             ;YES
        CPI     0BH             ;IS IT HOME
        JZ      HOM             ;YES
        CPI     0CH             ;IS IT FF
        JZ      CLEAR           ;YES
        JMP     EXIT            ;NOT WANTED-DO NOTHING
CRTN:
        MOV     A,L             ;GET CURSOR LOC LSB
        ANI     0C0H            ;PUT AT BEGIN OF LINE
        MOV     L,A             ;PUT BACK
        JMP     EXIT            ;GET OUT
DELETE:
        MOV     A,L             ;GET CURSOR LOC LSB
        ANI     61              ;SEE IF BEGIN OF LINE
        JZ      EXIT            ;YES - DO NOTHING
        DCX     H               ;
        MVI     M,' '           ;SET NEW LOC TO SPACE
        JMP     EXIT            ;GET OUT
CLF:
        PUSH    H               ;SAVE REGS FOR SCROL
        PUSH    B               ;
        LXI     B,64            ;PUT LINE LENGTH IN BC
        DAD     B               ;ADD TO CURRENT LOC
        MOV     A,H             ;GET LOC MSB
        CPI     VDBAS2+4        ;IS IT PAST END OF SCREEN
        POP     B               ;RESTORE BC
        JNZ     XTRA            ;HL OK BUT GIGO ON STACK
        CALL    SCRL            ;SCROLL IT
        POP     H               ;RESTORE HL
        JMP     EXIT            ;GET OUT
XTRA:
        XTHL                    ;SWAP HL WITH TOS
        POP     H               ;CLEAN STACK AND CORRECT HL
        JMP     EXIT            ;GET OUT
CLEAR:
        LXI     H,VDBASE        ;GET START OF SCREEN IN HL
        MVI     A,VDBAS2+4      ;PUT END OF SCREEN MSB IN A
CLEAR1:
        MVI     M,' '           ;PUT BLANK IN MEM
        INX     H               ;POINT TO NEXT LOC
        CMP     H               ;CHECK FOR END OF SCREEN
        JNZ     CLEAR1          ;NO, STAY IN LOOP
HOM:
        LXI     H,VDBASE        ;PUT START OF SCREEN IN HL
        JMP     EXIT            ;
EXIT:
        MOV     A,H             ;GET MEMORY BYTE
        ORI     80H             ;TURN ON CURSOR
        MOV     H,A             ;PUT IT BACK
        MOV     A,C             ;MAKE IT LOOK NORMAL
        RET                     ;GOODBYE
;
VDMP    DW      0               ;CURRENT CURSOR LOC
;
        ENDIF                   ;
;
;WRITE A CHARACTER ON LISTING DEVICE
;
LIST:
        IF      MYSYS           ;
        MVI     A,0AH           ;CHECK FOR A CR
        CMP     C               ;IF IT IS THEN DO NULL RTN
        MVI     A,7F            ;
        JZ      ADDNL           ;
        MVI     A,0CH           ;CHECK FOR FF
```

```
        CMP     C               ;DO FF WITH IF YES
        MVI     A,25            ;
        JZ      ADDNL           ;
        MVI     A,0DH           ;
        CMP     C               ;
        MVI     A,02            ;
        JZ      ADDNL           ;
LIST:
        IN      2               ;CHECK STATUS
        ANI     40H             ;MASK OFF TBE
        JZ      LIST            ;WAIT FOR TBE
        MOV     A,C             ;GET DATA BYTE
        OUT     01              ;SEND IT OUT
        RET                     ;
ADDNL:  PUSH    B               ;SAVE BC
        MOV     B,A             ;SAVE IT IN B
ADDNL1: CALL    LIST1           ;PRINT CR FIRST
ADDNL3: MVI     C,07FH          ;GET NULL CHAR
        DCR     B               ;DECREMENT COUNTER
        JNZ     ADDNL1          ;IF <>0 THEN DO MORE NULLS
ADDNL4: POP     B               ;RESTORE B
        MOV     A,C             ;RESTORE A
        RET                     ;RETURN FROM LIST
        ENDIF                   ;
;
;
        IF      NOT MYSYS       ;
        CALL    CONOT           ;ROUTE TO CONSOLE FOR NOW
        RET                     ;
        ENDIF                   ;
;
;NORMALLY USED TO PUNCH PAPER TAPE
;CAN BE USED AS BIT BUCKET TO CHECK FILES
;
PUNCH:
        RET                     ;
;
;NORMALLY USED TO READ PAPER TAPE
;NOT IMPLEMENTED BUT CPM REQUIRES EOF
;
READER:
        MVI     A,1AH           ;SET A=CTL-Z (EOF)
        RET                     ;
;
;DATA AREAS FOR CPM
;
TRK     DB      0               ;TRK WANTED
TRKTB   DB      1,1,1,1         ;TRACK TABLE
SECT    DB      0               ;SECTOR #
DMAADD  DW      0               ;DMA ADDRESS
DISKNO  DB      0               ;SELECTED DISK
NXTDSK  DB      0               ;
HOLD    DB      0               ;SAVE AREA FOR SEEK
;
ERCNT:  DB      0               ;ERROR COUNT
ERRS:   DB      0               ;ERROR HOLD AREA
SKCNT:  DB      0               ;SEEK ERROR HOLD
;
;
        END
```

Cold Boot Automatic Program Load and Execute

Lorin S. Mohler

The following is a method for modifying your systems BIOS to allow assembly of a system for automatic program loading on a cold boot. Descriptions are preceeded by an asterisk (*) and, of course, are not part of the BIOS modification.

The AUTO switch may be set true to produce the code needed for CCP to automatically execute a single command line directly after the initial system load. A suggested command to use is SUBMIT INIT, which requires that only SUBMIT.COM and a submit file INIT.SUB be present on the drive A. This way, by controlling what is in the submit file, multiple as well as single commands can be executed after system loading.

```
*assembly switch to enable or disable AUTO
*someplace above here TRUE and FALSE must be defined.
*I like
*FALSE EQU 0
*TRUE EQU NOT FALSE
AUTO SET TRUE                ; If auto start SUBMIT INIT
*CPMB is the first code location of the CCP portion of BDOS
*this remains as if for your BIOS and is given here to show that
*it preceeds the AUTO code
CPMB EQU (MSIZE*1024)-xx  ; system origin

*The following is the AUTO code that patches the CCP
        IF AUTO          ; auto start-up feature
        ORG CPMB +       ; start patching CCP here
        DB ACLEN         ; message length calculated later
ACMSG: DB  'SUBMIT INIT' ; command line to be executed
ACLEN: EQU  $-ACMSG      ; message length calculation
       ENDIF
```

Lorin S. Mohler, P.O. Box 8340, Anaheim, CA 92802.

```
*The BIOS is now ORGed
*This remains as is in your BIOS and is shown for reference
BIOS ORG CPMB + ...          ; BIOS org for system generation

*The following code is added in the WARM BOOT code.
*Its function is to turn off the automatic program load
*operation so a warm boot ( C) will not initiate another AUTO
*Sequence. If you do not put this code in, you will not be able
*to get back to the promptA .

        IF AUTO
        XRA A
        STA CPMB +     ;set command line empty
        ENDIF
```

After the BIOS is edited and assembled the standard system generation procedure is used. Refer to your documentation. For most of us, it looks like:

```
DDT CPMxx.COM  ;load DDT and CPM of the appropriate size
IBIOS.HEX
Rxxxx          ;refer to your documentation
               ;the patches now overlay the CCP
IBOOT.HEX
R900           ;load the booter
GO             ;exit DDT
SYSGEN
               ;write the system to diskette
etc. . .
```

When the diskette is cold booted (RESET) the normal sign-on message should appear, then the AUTO command line will be executed. This, for example, may take you directly into a word processor or MBasic or whatever is in the INIT.SUB file. Let me know how useful this is to you and what your application is.

Choosing Between CRT Output and Printer Output

Bob Kowitt

Some versions of Basic allow you to specify while running your program whether you want to output to your CRT terminal or to your printer. Unfortunately, one of the most widely used and powerful Basics, Microsoft Basic, does not. If you use the methods proposed in the user's manual, you are told to use the command PRINT to go to the CRT terminal and the command LPRINT when you want to output to your printer.

There is, however, a way you can bypass this deficiency if you are using Microsoft Basic Rel. 5.0 or later, under CP/M. Locations 0000,0001, and 0002 contain the jump to the BIOS in CP/M. Microsoft Basic uses the data stored at these locations to direct your output as you have chosen with the commands PRINT or LPRINT in your program. Using this same information, you can locate the point in memory that contains your routine to write to the terminal or to the printer.

You can bypass the use of LPRINT by fooling the Microsoft interpreter. In Microsoft BASIC 5.0 and higher, this data is stored at a location between 16000 and 18000 (decimal), depending on which release you are using. The location changed during modification of Microsoft Basic to eliminate bugs that were discovered after the original release. By including within your program the following routine, you can at any time within your program direct the output in either direction at runtime rather than being forced to duplicate the code when writing your program. You cannot poke the data directly into the jump table of CP/M because Microsoft Basic does not use this jump table after finding its location.

Lines 60 to 100 define your variables and prepare your program for further input during your program.

Poke F, OT (line 160) should be inserted before each point at which you may want to change the output. Poke F,C (line 180) should be inserted to get output back to your CRT terminal. You must put a copy of line 180 at the end of your program. If you don't, you will be locked into your printer and not your CRT at the end of the program. Your keyboard will still be entering data to your computer but there will be output to the printer and not the CRT.

Bob Kowitt, 1727 N. Jerusalem Rd., East Meadow, NY 11554.

```
10      '       **************
20      '       SAMPLE PROGRAM
25      '       **************
26      '
27      'To operate printer, fill F with PRINTBYTE
30      'To operate console, fill F with CONSOLEBYTE
31 BIOSBOTTOM=(PEEK(2)*256)+PEEK(1)
32 PRNTBYTELOC=BIOSBOTTOM+13 : CONSOLEBYTELOC=BIOSBOTTOM+10
33 PRNTBYTE=PEEK(PRNTBYTELOC) : CONSOLEBYTE=PEEK(CONSOLEBYTELOC)
34 C=CONSOLEBYTE         '       F is location with MBASIC that directs
35 FOR I=16000 TO 18000
36 IF PEEK(I)=CONSOLEBYTE AND PEEK(I+1)=PEEK(CONSOLEBYTELOC+1) THEN 38
37 NEXT
38      '
110     '
120 INPUT "Do you want P(rinter)  or C(onsole)  ";CHOICE$
130 OT=C
140 IF LEFT$(CHOICE$,1)="P" THEN OT=PRNTBYTE
150     '
160 POKE F,OT
170 PRINT"This is a demonstration of print output selection"
180 POKE F,C
190 END
```

Should you get trapped in printer mode, simply type:

POKE F,C (cr)

to regain control and printout at the console.

Dot Graphics on the IMSAI — VIO

Gary Sabot

Run TRS-80, Apple or PET graphics programs on your IMSAI-VIO and similar video boards with this program.

Several of the personal computers in wide use today are capable of displaying low resolution graphics. This means that they are able to display dots, lines, pictures, or even animated characters on their screens. The TRS-80 and the Apple have this capability. Because of the widespread use of these two computers, there are many programs available which make use of their graphics capability. This article presents a program that will enable owners of the Imsai VIO (or similar memory-mapped displays, such as the Polymorphics VTI) to utilize these programs. By making a few simple changes, *all* TRS-80 graphics programs, most Apple graphics programs (those programs which do not make extensive use of color), and some PET graphics programs will run on your machine.

The Imasi VIO is capable of displaying special "graphics characters." (See Figure 1.) Each graphics character contains six squares. By using the proper graphics character, it is possible to turn each of these six squares on (white), or off (black) independently. The problem is: how can a single square be turned on, without disturbing the five squares that surround it?

My solution to this problem is in the form of a machine language program. (See listing #1.) It allows a Basic program to quickly and easily plot points using the VIO. It can be modified to work with other memory-mapped display boards, such as the Polymorphics VTI. The program is designed to be used in conjunction with Microsoft Basic and CP/M. (Of course, it can also be used by a machine language program.)

To plot a point, the proper graphics character must be selected; then this character must be placed in the correct memory location. If this process were to be implemented as a Basic program, it would take approximately one-half second to plot each point. If a program that plots several hundred points were run, however, those one-half seconds would add up, delaying the program. I have implemented the plotting program in machine language, because a machine language program is considerably faster than an equivalent program written in Basic. If a Basic program needs to turn a certain square "on," it simply passes its X-Y coordinates to this routine (see Figure 2) and calls it, using the USR function. The routine then turns the square "on," and subsequently returns to the Basic program. Analagous procedures may be used to determine the square's present color (black or white), or to turn it off (black).

Gary Sabot, 38 Woodland Rd., Roslyn, NY 11576.

Utilizing The Program

If you have a 30K CP/M system using the Imsai VIO, you can employ the program just as I assembled it. To use the routine (after you have POKE'd it into memory—see the Basic listing), first POKE the Y coordinate of the desired pixel into location 6889H, then POKE the X coordinate into 688AH, and POKE the function number into 688BH. The function number would be a 1 to set the pixel white. This is equivalent to the TRS-80's SET(X,Y) command. The function number would be a 0 to set the pixel black. This is identical to the TRS-80's RESET(X,Y) command. If the function number is a two, the plotting routine will determine the present status of the pixel, without disturbing it. This is similar to the TRS-80's POINT(X,Y) command. The status of the pixel is retrieved by a PEEK to 688BH. A 0 will be found there if

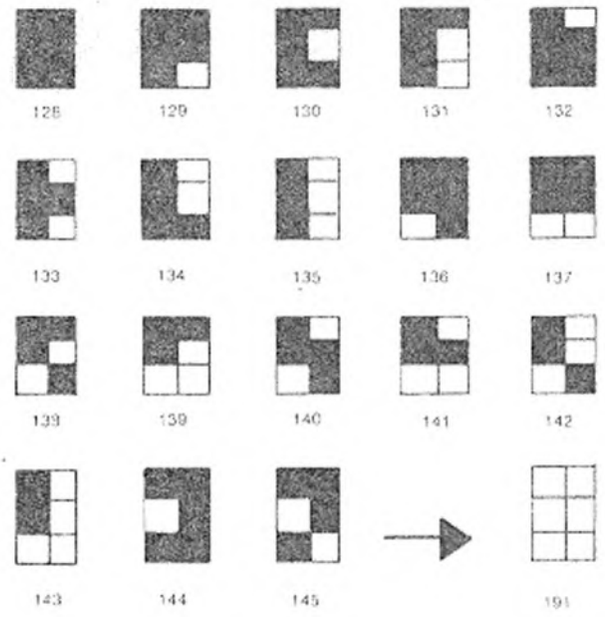


Figure 1. Some of the Imsai VIO's "graphic characters." The number of the desired character can be placed in the VIO's refresh memory; subsequently, the character will appear on the screen.

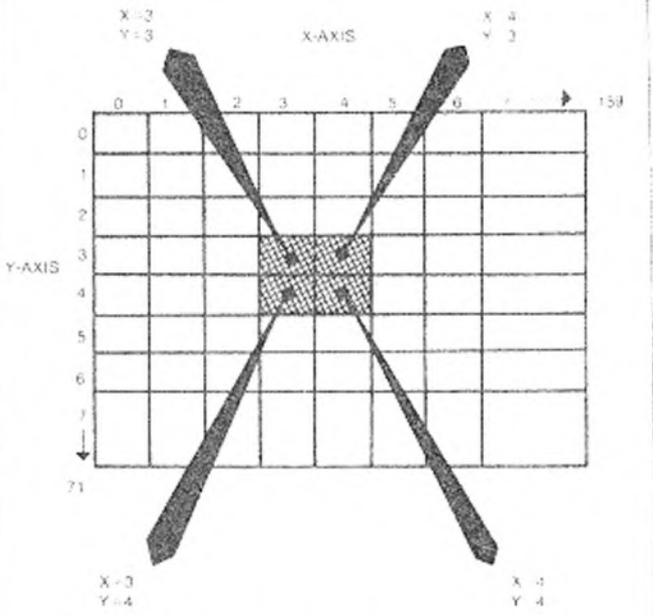


Figure 2. This is the coordinate system used to address the squares, or pixels, on the screen. It is the same as the system used by the TRS-80, except that the VIO's display is larger (160 by 72, compared with the TRS-80's 128 by 48).

the pixel is black and a 1 will be found there if the pixel is white. Once the proper values for the X,Y coordinates, and the function number have been POKED into memory, jump to 6800H using the USR function to execute the plotting routine.

I have provided a sample Basic program which uses my point plotting routine. (See listing #2.) It is a modification of program on page 33 of *Introduction to Low Resolution Graphics*, by Nat Wadsworth. The program draws lines between random points on the screen.

This plotting routine enables you to adapt to your computer the many TRS-80, Apple, and PET programs employing graphics which are in the public domain.

Because the plotting program is written in machine language, it is necessary to reserve memory for it when MBasic is run. To do this, instead of simply typing "MBasic" to run MBasic, type "MBasic /M:&H67FF". This sets 67FF as the highest address available for use by MBasic. The preface "&H" indicates that a hex number will follow.

If your system is larger than 30K, you might want to reassemble the routine at a higher address, in order to fully utilize your memory. For every kilobyte of memory that you have above 30K, add 400H to 67FFH. Then reassemble the routine at the resulting address. Finally, substitute the new origin for the 67FFH in "MBasic /M:&H67FF". Of course, the locations for POKING and PEEKING change each time a different version of this program is assembled.

If you are using a video board other than the VIO, you will probably have to reassemble the program, making one or more of the following changes (most involve the "EQUates" in the beginning of the program):

1) If your video board is addressed at a location other than 61440 (F000H), change the value in the line beginning "SCREENAD EQU..." in the listing to the correct screen address.

2) If the line length of your display is different than 80 characters per line, change the line length in the listing ("LINE EQU...") to the proper value, and reassemble it.

3) If a black pixel is represented by a 1, and not a 0, substitute 1 for 0 in the listing where it now reads "BLK EQU 0".

4) Determine the value of a blank (all black) character cell and substitute this for the 80H in the line "BLKCHR EQU 80H".

5) Find out what the proper CHRAND and CHRCPL values for your display should be (refer to the comments in the program listing) and insert the correct values into the program.

6) If the progression from black to white on your video board is different than that shown in Figure 1, you will have to modify the portion of the program (6843H thru 6863H) that calculates the bit mask. This can be a very involved process!

Now that your computer has dot graphics capability, make it work for you. Programs can be written with output in the form of a graph, instead of in numbers and letters. Programs can even be designed to imitate arcade games. One of the most exciting possibilities of this plotting routine is that it enables you to adapt to your computer the many TRS-80, Apple, and PET programs employing graphics which are in the public domain.

LISTING #1

```
                        ;******* POINT PLOT ROUTINE FOR MEMORY MAPPED DISPLAYS *******
                        ;******** BY GARY SABOT 11/27/79 ********
F000 =          SCREENAD        EQU     61440   ;ADDRESS OF VIDEO BOARD
0050 =          LINE            EQU     80      ;LINE LENGTH
0000 =          BLK             EQU     0       ;BIT THAT REPRESENTS A
                                                ;PIXEL THAT IS "OFF" (BLACK)
0080 =          BLKCHR          EQU     80H     ;CONTENTS OF A BLANK (BLACK)
00C0 =          CHRAND          EQU     0C0H
0080 =          CHRCPI          EQU     80H
                ;CHRAND IS "ANDED" WITH THE CONTENTS OF A CHARACTER CELL.  IF
                ;THE CELL CONTAINS A VALID GRAPHICS CHARACTER (NON-ALPHABETIC)
                ;THE RESULT SHOULD EQUAL CHRCPI.  IF IT DOES NOT, A BLANK
                ;CHARACTER WILL BE PLACED IN THE CELL.
6800                    ORG     6800H
                ;
                ;DATA SHOULD BE STORED IN FORM Y, X, FUNCTION #
                ;FUNCTION #-- 0 MEANS SET BLACK, 1 MEANS SET WHITE,
                ;ALL ELSE MEANS RETURN DOT STATUS (0=BLACK, 1=WHITE)
                ;
6800 218968             LXI     H,DATA          ;SET POINTER TO DATA
6803 0E00               MVI     C,0             ;LOAD C WITH ZERO
6805 7E                 MOV     A,M             ;LOAD Y INTO A
6806 D603       DIV:    SUI     3               ;DIVIDE Y BY 3,
                                                ;PUT RESULT IN C
6808 DA1068             JC      DIVX
680B 0C                 INR     C
680C A7                 ANA     A
680D C20668             JNZ     DIV
6810 328C68     DIVX:   STA     YREM            ;SAVE REMAINDER FROM
                                                ;DIVISION OF Y
6813 23                 INX     H               ;READ IN VALUE OF X
6814 7E                 MOV     A,M
6815 0F                 RRC                     ;DIVIDE X BY 2
6816 E67F               ANI     7FH             ;STRIP OFF FIRST BIT
6818 2100F0             LXI     H,SCREENAD      ;LOAD ADDRESS OF VIDEO BOARD
681B 5F                 MOV     E,A             ;CALCULATE CHARACTER'S
                                                ;LOCATION IN MEMORY
681C 1600               MVI     D,0
681E 19                 DAD     D               ;ADD X TO BASE ADDRESS
681F 115000             LXI     D,LINE          ;PREPARE TO MULTIPLY
6822 79                 MOV     A,C
6823 A7                 ANA     A
6824 CA3068             JZ      ADFND
6827 3D         MULT:   DCR     A               ;ADD Y*LINE LENGTH TO
                                                ;BASE ADDRESS
6828 CA2F68             JZ      ONEMOR          ;PUT ADDRESS IN HL
682B 19                 DAD     D
682C C32768             JMP     MULT
682F 19         ONEMOR: DAD     D
                ;
                ADFND:
                ;FIRST CHECK IF A GRAPHICS CHARACTER IS ALREADY IN CHARACTER
                ;CELL.  IF NOT, STORE A BLACK CHARACTER THERE.
                ;THEN CALCULATE BIT MASK
6830 7E                 MOV     A,M             ;LOAD CHARACTER
6831 E6C0               ANI     CHRAND          ;CHECK IF GRAPHICS CHARACTER
6833 FE80               CPI     CHRCPI
6835 CA3B68             JZ      FNDBIT          ;YES, NOW CONTINUE
6838 3E80               MVI     A,BLKCHR        ;NO, STORE AN ALL
683A 77                 MOV     M,A             ;BLACK CHARACTER
683B 3A8A68     FNDBIT: LDA     DATA+1          ;LOAD VALUE OF X
683E E601               ANI     1               ;GET REMAINER FROM
                                                ;DIVISION BY 2
6840 CA4868             JZ      XRZER           ;REMAINDER IS ZERO
6843 3E01               MVI     A,1             ;BEGIN TO FORM BIT MASK IN A
6845 C34A68             JMP     CONT            ;CONTINUE
6848 3E08       XRZER:  MVI     A,8
684A 47         CONT:   MOV     B,A             ;SAVE PARTIALLY FORMED MASK
684B 3A8C68             LDA     YREM            ;LOAD REMAINER FROM
                                                ;DIVISION OF Y
684E A7                 ANA     A               ;SET FLAGS
684F CA6068             JZ      SHIFT2          ;NEEDS TO BE SHIFTED TWICE
6852 3C                 INR     A
6853 CA6468             JZ      MSKFND          ;DOES NOT NEED TO BE SHIFTED
6856 3C                 INR     A
6857 C26068             JNZ     SHIFT2          ;MUST BE SHIFTED TWICE
685A 78         SHIFT1: MOV     A,B             ;LOAD BIT MASK
685B 07                 RLC                     ;SHIFT LEFT ONCE
685C 47                 MOV     B,A             ;SAVE IN B
685D C36468             JMP     MSKFND
6860 78         SHIFT2: MOV     A,B             ;LOAD BIT MASK
6861 0707               RLC ! RLC               ;SHIFT LEFT TWICE
6863 47                 MOV     B,A             ;SAVE IN B
                ;
                ;THE BIT MASK IS IN B AND THE CHARACTER ADDRESS IS IN HL.
                ;NOW COMPUTE AND PUT PROPER VALUE ON THE SCREEN.
                MSKFND:
6864 3A8B68             LDA     DATA+2          ;GET FUNCTION #
6867 A7                 ANA     A
6868 CA7E68             JZ      SETBLK          ;0 MEANS SET DOT BLACK
686B 3D                 DCR     A
686C CA8568             JZ      SETWHT          ;1 MEANS SET DOT WHITE
                ;
                ;ASSUME CALLER WANTS DOT STATUS
                ;
686F 7E                 MOV     A,M             ;LOAD CHARACTER CELL
6870 A0                 ANA     B               ;AND WITH BIT MASK
                        IF      BLK
                        JNZ     RETBLK
                        ENDIF
                        IF      NOT BLK
6871 CA7A68             JZ      RETBLK
                        ENDIF
6874 3E01               MVI     A,1             ;RETURN A 1 FOR "WHITE"
6876 328B68             STA     DATA+2
6879 C9                 RET
687A 328B68     RETBLK: STA     DATA+2          ;RETURN A 0 FOR "BLACK"
687D C9                 RET


                ;THE FOLLOWING ROUTINES ALL LOAD THE CHARACTER CELL,
                ;MODIFY IT IN THE DESIRED WAY, SAVE IT, AND RETURN.
                ;
                ;SET DOT BLACK
                ;
                SETBLK:
                        IF      BLK             ;CONDITIONAL ASSEMBLY
                                                ;WHEN BLACK=1
                        MOV     A,M
                        ORA     B
                        MOV     M,A
                        RET
```

```
                        ENDIF
                ;
                        IF      NOT BLK
687E 78                 MOV     A,B
687F 2F                 CMA
6880 47                 MOV     B,A
6881 7E                 MOV     A,M
6882 A0                 ANA     B
6883 77                 MOV     M,A
6884 C9                 RET
                        ENDIF
                ;
                ;SET DOT WHITE
                ;
                SETWHT:
                        IF      BLK
                        MOV     A,B
                        CMA
                        MOV     B,A
                        MOV     A,M
                        ANA     B
                        MOV     M,A
                        RET
                        ENDIF
                ;
                        IF      NOT BLK
6885 7E                 MOV     A,M
6886 B0                 ORA     B
6887 77                 MOV     M,A
6888 C9                 RET
                        ENDIF

6889            DATA    DS      3               ;STORAGE FOR Y,X,
                                                ;AND FUNCTION NUMBER
688C            YREM    DS      1               ;STORAGE FOR REMAINER OF Y/3
688D                    END     6830H
```

An 8080 Disassembler

William Yarnall

A Disassembler program will take object code, in RAM, and yield a source code output. It is an invaluable software utility. This Disassembler program takes up only 1K of memory space and is very easy to use.

This is a very simple and easy to use disassembler program that can be used with virtually any 8080 microcomputer system. Previous to my getting disks, I used it with my audio cassette based system. I now use it on my IMSAI with North Star Disk System, in both CP/M and North Star DOS.

I have several versions on disk that are assembled to run at different memory locations. For North Star DOS I have versions that are assembled to run at 0, 2A00H, 5800H and D000H. For CP/M I have versions assembled to run at 100H, 5800H and D000H. I then load the appropriate version for the software I am going to disassemble. For example, to disassemble North Programs that have an origin of 2A00H, I load and run the disassembler at memory location 0. To disassemble a CP/M program that has an origin at 100H, I load and run the disassembler in high memory (e.g. 5800H or D000H depending on where CP/M is located).

William Yarnall, ACGNJ, 1176 Raritan Road, Scotch Plains, NJ 07076.

The version of disassembler shown here is set up to run with my monitor program that starts at E000H. Therefore the program starts with "global definitions" that define the locations of the console (keyboard and VDM display in my case) I/O driver routines and the return address for the monitor or operating system. These equates should therefore be changed to suit your particular DOS or monitor program.

Routine LOOP, near the beginning of the program reads the sense switches on the front panel of my IMSAI-8080. If switch 2 is raised then the disassembly is interrupted and control is returned to the operating system. This allows me to interrupt a run at any point before the run is completed, should I desire to do so. If your system does not have a front panel with sense switches and you want to be able to interrupt a run then replace the IN 0FFH and ANI 2 instructions with a CALL to the keyboard status routine followed by a jump to the operating system. Then any keypress will interrupt execution and return to the operating system. This works if the keyboard input routine returns with a non-zero. If it returns a zero change JNZ EEND to JZ EEND.

To use the disassembler program load it into memory and then type the starting address of the memory area to be disassembled followed by a comma and the ending address to be disassembled.

```
A>DIR
A: NSTAT    COM
A: ZDIR     COM
A: ED       COM
A: DISASM   PRN
A: DISASM   ASM
A>TYPE DISASM.ASM
; GLOBAL DEFINITIONS
INP     EQU     0E092H  ; KEYBOARD INPUT
OUT8    EQU     0E17CH  ; CHARACTER DISPLAY
EEND    EQU     0E02BH  ; RETURN TO MONITOR
        ORG     0
;
; MAIN PROCESSING LOOP
; GET STARTING ADDRESS
        LXI     H,0
GIN0:   PUSH    H
        CALL    INP     ; GET KEYBOARD CHAR
        CALL    OUT8    ; ECHO
        CALL    NYBLE
        POP     H
        JC      GIN1
        MOV     E,A
```

```
        MVI     D,0
        DAD     H
        DAD     H
        DAD     H
        DAD     H
        DAD     D
        JMP     GIN0
GIN1:   XRA     A
        XCHG
LOOP:   IN      0FFH    ; SENSE SWITCH
        ANI     2       ; NO. 2 TO QUIT
        JNZ     EEND    ; RETURN TO MONITOR
        CALL    CLER
        CALL    PROC
        CALL    DISP
        JMP     LOOP
; CONVERT ASCII TO HEX
NYBLE:  CPI     '0'
        RC
        CPI     'F'+1
        CMC
        RC
        SUI     '0'
        CPI     10
        CMC
        RNC
        SUI     7
        CPI     10
        RET
; ADD (A) TO H,L
ADDH:   ADD     L
        MOV     L,A
        RNC
        INR     H
        RET
; GET AND STORE MNEMONIC
;  (HL) POINTS TO MNEMONIC TABLE
CARD:   PUSH    H
        MVI     A,' '
CRD0:   STA     PMNE+3
        MOV     A,M
        ORA     A
        JZ      CRD2
        MOV     B,A
        RRC
        RRC
        RRC
        ANI     1FH
        ADI     40H
        STA     PMNE
        INX     H
        MOV     C,M
        MOV     A,B
        ANI     7
        RLC
         LC
        MOV     B,A
        MOV     A,C
        RLC
        RLC
        ANI     3
        ORA     B
        ADI     40H
        STA     PMNE+1
        MOV     A,C
        ANI     1FH
        JZ      CRD1
        ADI     40H
        STA     PMNE+2
CRD1:   POP     H
        INX     H
        INX     H
        RET
; 4-CHAR MNEMONIC
CRD2:   INX     H
        MOV     A,M
        LXI     H,TBX
        CALL    ADDH
        MOV     A,M
        INX     H
        JMP     CRD0
; CHECK FOR REG "PSW"
CHEK:   CPI     0F1H
        JZ      CHEK0
        CPI     0F5H
        RNZ
CHEK0:  LXI     H,'SP'
        SHLD    PARG
        MVI     A,'W'
        STA     PARG+2
        RET
; INITIALIZE OUTPUT BUFFER
CLER:   LXI     H,PLOC
        MVI     B,36
CLER0:  MVI     M,' '
        INX     H
        DCR     B
        JNZ     CLER0
        MVI     M,13
; CONVERT & STORE PC
        PUSH    D
        POP     H
        LXI     B,PLOC
        CALL    C4D
; CONVERT & STORE COMMAND
        LDAX    D
        MOV     H,A
        LXI     B,PG1
        CALL    C2D
        RET
; GET AND STORE ARGUMENT
;  (HL) POINTS TO PROCESS BYTE
CREG:   MOV     A,M
        RLC
        JNC     CR1
; REGISTER PAIR ARGUMENT
        PUSH    PSW
        LDAX    D
        ANI     30H
        RRC
        RRC
        RRC
        RRC
        LXI     H,REG1
        CALL    ADDH
        MOV     A,M
        STA     PARG
        LDAX    D
        CALL    CHEK
        POP     PSW
CR1:    RLC
        JNC     CR3
; SINGLE REGISTER ARGUMENT
        PUSH    PSW
        LDAX    D
        CPI     40H
        JNC     CR6
        ANI     38H
        RRC
        RRC
        RRC
```

```
CR2:    LXI     H,REG2
        CALL    ADDH
        MOV     A,M
        STA     PARG
        POP     PSW
CR3:    RLC
        JNC     CR7
; 1-BYTE CONSTANT ARGUMENT
        INX     D
        LDAX    D
        MOV     H,A
        PUSH    H
        LXI     B,PARG+2
        CALL    C2D
        POP     H
CR3A:   LXI     B,PC2
        CALL    C2D
CR4:    LDA     PARG
        CPI     ' '
        JZ      CR5
        MVI     A,','
        STA     PARG+1
CR5:    CALL    JUST
        RET
CR6:    ANI     7
        JMP     CR2
CR7:    RLC
        JNC     CR8
; 2-BYTE CONSTANT ARGUMENT
        INX     D
        LDAX    D
        MOV     L,A
        INX     D
        LDAX    D
        MOV     H,A
        PUSH    H
        LXI     B,PARG+2
        CALL    C4D
        POP     H
        LXI     B,PC3
        CALL    C2D
        JMP     CR3A
CR8:    LDAX    D
; TEST FOR "RST" AS SPECIAL CASE
        ANI     0C7H
        CPI     0C7H
        JNZ     CR5
        LDAX    D
        ANI     38H
        RRC
        RRC
        RRC
        ADI     '0'
        STA     PARG
        RET
; LEFT JUSTIFY ARGUMENT FIELD
JUST:   LDA     PARG
        CPI     'S'
        JZ      JUS2
        CPI     ' '
        RNZ
        LDA     PARG+2
        CPI     ' '
        RZ
; LEFT SHIFT FIELD BY 2
        PUSH    B
        PUSH    D
        PUSH    H
        PUSH    PSW
        LXI     H,PARG
        LXI     D,PARG+2
        MVI     B,6
JUS0:   LDAX    D
        MOV     M,A
        INX     H
        INX     D
        DCR     B
        JNZ     JUS0
JUS1:   POP     PSW
        POP     H
        POP     D
        POP     B
        RET
; MAKE PLACE FOR 1 ADD'L CHAR.
JUS2:   PUSH    B
        PUSH    D
        PUSH    H
        PUSH    PSW
        LXI     H,PARG+6
        LXI     D,PARG+5
        MVI     B,5
JUS3:   LDAX    D
        MOV     M,A
        DCX     D
        DCX     H
        DCR     B
        JNZ     JUS3
        MVI     A,'P'
        MOV     M,A
        JMP     JUS1
; CONVERT 1 BYTE TO 2 ASCII CHARS.

C2D:    PUSH    D
        MVI     D,2
        JMP     CVD
; CONVERT 2 BYTES TO 4 ASCII CHARS.
C4D:    PUSH    D
        MVI     D,4
CVD:    XRA     A
        DAD     H
        RAL
        DAD     H
        RAL
        DAD     H
        RAL
        DAD     H
        RAL
        CPI     10
        JC      CVD0
        ADI     7
CVD0:   ADI     '0'
        STAX    B
        INX     B
        DCR     D
        JNZ     CVD
        POP     D
        RET
; DISPLAY LINE
DISP:   PUSH    H
        LXI     H,PLOC
DISP0:  MOV     A,M
        CALL    OUT8
        CPI     13
        JZ      DISP1
        INX     H
        JMP     DISP0
DISP1:  POP     H
        RET
; PROCESS COMMAND (DE POINTS TO COMMAND)
PROC:   PUSH    D
```

```
        LDAX    D
        CPI     40H
        JNC     PRB
; COMMAND 00 - 3F
        ANI     7
        RLC
        MOV     C,A
        RLC
        ADD     C
        LXI     H,PRA
        CALL    ADDH
        PCHL
PRA:    LXI     H,TB1
        JMP     PRA4
        LXI     H,TB2
        JMP     PRA5
        LXI     H,TB3
        JMP     PRA1
        LXI     H,TB4
        JMP     PRA5
        LXI     H,TB5
        JMP     PRA3
        LXI     H,TB6
        JMP     PRA3
        LXI     H,TB7
        JMP     PRA3
        LXI     H,TB8
PRA1:   LDAX    D
        ANI     38H
        RRC
        RRC
        MOV     C,A
        RRC
        ADD     C
PRA2:   CALL    ADDH
PRA3:   CALL    CARD
        POP     D
        CALL    CREG
        INX     D
        RET
PRA4:   LDAX    D
        ORA
        JZ      PRA3
        MVI     A,3
        JMP     PRA2
PRA5:   LDAX    D
        ANI     8
        JZ      PRA3
        MVI     A,3
        JMP     PRA2
PRB:    CPI     80H
        JNC     PRC
; COMMAND 40 - 7F
        LXI     H,TB9
        LDAX    D
        CPI     76H
        JZ      PRA3
; MOV INSTRUCTION
        LXI     H,'OM'
        SHLD    PMNE
        LXI     H,' V'
        SHLD    PMNE+2
        ANI     38H
        RRC
        RRC
        RRC
        LXI     H,REG2
        CALL    ADDH
        MOV     L,M
        MVI     H,','
        SHLD    PARG
        LDAX    D
        ANI     7
        LXI     H,REG2
        CALL    ADDH
        MOV     A,M
        STA     PARG+2
        POP     D
        INX     D
        RET
PRC:    CPI     0C0H
        JNC     PRD
; COMMAND 80 - BF
        LXI     H,TB10
        JMP     PRA1
; COMMAND C0 - FF
PRD:    SUI     0C0H
        MOV     C,A
        RLC
        ADD     C
        LXI     H,TB11
        JMP     PRA2
; REGISTER TABLES
REG1:   DB      'B'
        DB      'D'
        DB      'H'
        DB      'S'
REG2:   DB      'B'
        DB      'C'
        DB      'D'
        DB      'E'
        DB      'H'
        DB      'L'
        DB      'M'
        DB      'A'
; COMMAND PROCESSING TABLES
;  CONTAINS MNEMONICS & PROCESSING FLAGS
TB1:    DW      0D073H
        DB      0
        DW      922CH
        DB      0
TB2:    DW      966H
        DB      90H
        DW      4420H
        DB      80H
TB3:    DW      0
        DB      80H
        DW      300H
        DB      80H
        DW      0
        DB      80H
        DW      300H
        DB      80H
        DW      600H
        DB      10H
        DW      900H
        DB      10H
        DW      19DH
        DB      10H
        DW      161H
        DB      10H
TB4:    DW      984BH
        B       80H
        DW      0D820H
        DB      80H
TB5:    DW      924BH
        DB      40H
TB6:    DW      0D220H
        DB      40H
TB7:    DW      896DH
        DB      60H
TB8:    DW      393H
```

```
        DB      0               DB      0
        DW      8394H           DW      0F1CH
        DB      0               DB      10H
        DW      4C90H           DW      0C00H
        DB      0               DB      80H
        DW      5290H           DW      890BH
        DB      0               DB      20H
        DW      4120H           DW      0D494H
        DB      0               DB      0
        DW      411BH           DW      594H
        DB      0               DB      0
        DW      39DH            DW      1B00H
        DB      0               DB      0
        DW      431BH           DW      554H
        DB      0               DB      10H
TB9:    DW      1443H           DW      1500H
        DB      0               DB      0
TB10:   DW      409H            DW      51CH
        DB      40H             DB      10H
        DW      309H            DW      922CH
        DB      40H             DB      0
        DW      429DH           DW      89C4H
        DB      40H             DB      20H
        DW      8298H           DW      0D494H
        DB      40H             DB      0
        DW      810BH           DW      94H
        DB      40H             DB      0
        DW      81C4H           DW      0D083H
        DB      40H             DB      80H
        DW      817CH           DW      54H
        DB      40H             DB      10H
        DW      501BH           DW      547DH
        DB      40H             DB      20H
TB11:   DW      9A93H           DW      831BH
        DB      0               DB      10H
        DW      0D083H          DW      0C00H
        DB      80H             DB      80H
        DW      9A53H           DW      499DH
        DB      10H             DB      20H
        DW      5053H           DW      0D494H
        DB      10H             DB      0
        DW      9A1BH           DW      0C090H
        DB      10H             DB      0
        DW      0C00H           DW      922CH
        DB      80H             DB      0
        DW      909H            DW      0C050H
        DB      20H             DB      10H
        DW      0D494H          DW      804BH
        DB      0               DB      20H
        DW      8096H           DW      0C018H
        DB      0               DB      10H
        DW      5491H           DW      922CH
        DB      0               DB      0
        DW      8056H           DW      8998H
        DB      10H             DB      20H
        DW      922CH           DW      0D494H
        DB      0               DB      0
        DW      801EH           DW      0F94H
        DB      10H             DB      0
        DW      0F00H           DW      0D083H
        DB      10H             DB      80H
        DW      0C908H          DW      0F54H
        DB      20H             DB      10H
        DW      0D494H          DW      1200H
        DB      0               DW      4022H
        DW      8393H           DB      0
        DB      0               DW      1CH
        DW      0D083H          DB      10H
        DB      80H             DW      0C00H
        DW      8353H           DB      80H
        DB      10H             DW      897CH
        DB      20H             DW      0D494H
        DB      0               DW      4093H
        DB      0               DW      1800H
        DB      0               DW      4053H
        DB      10H             DW      402AH
        DB      0               DW      401BH
        DB      10H             DW      922CH
        DB      0               DW      91CH
        DB      20H             DW      0D494H
        DB      0
; 4-CHARACTER MNEMONIC TABLE
TBX:    DW      9D58H
        DB      1
        DW      6158H
        DB      1
        DW      9A44H
        DB      0CH
        DW      6244H
        DB      0CH
        DW      8548H
        DB      53H
        DW      184CH
        DB      4CH
        DW      0C54CH
        DB      8
        DW      0C047H
        DB      0C8H
        DW      9C4CH
        DB      8
        DW      804CH
        DB      0C8H
PLOC:   DS      5               ; OUTPUT BUFFER
PC1:    DS      3
PC2:    DS      3
PC3:    DS      3
PLAB:   DS      6
PMNE:   DS      6
PARG:   DS      10
        END
```

8080 Dynatrace

Charlie Foster & Richard Meador

A super 8080 emulator program useful in debugging 8080 programs.

Anyone who is learning to program in assembly language can use a method of observing just what is going on inside of the CPU. If you know how to program already, you still need a way to debug your new programs. Dynatrace will help you in either case. Dynatrace is a development tool that will accept commands from any standard ASCII keyboard and provides a TWO PART DISPLAY. The video monitor is configured for a 64 character by 16 line display but since this article includes the source any other configuration can be patched into the program.

The upper 4 lines of the display are dedicated to a dynamic display of the contents of register information being used in conjunction with the program being developed. The register display is always in view and is updated continuously as simulation progresses whether the simulation is single step or continuous run. Dynatrace is actually a pseudo-computer simulating in software everything done by the 8080 in hardware and more. With Dynatrace the user is able to see at a glance all register information and can make changes to existing contents of registers as desired.

Dynatrace is also easily reconfigured to utilize subroutines existing in a monitor the user may already have up and running. The A, E, F, H, J, K, N, O, P, Q, T, U, V, W, X and Y commands, being undefined, provide the user with the facilities for extensive expansion of the Dynatrace command set. Unused commands may be implemented by storing the address of a subroutine in the jump table beginning at address Base + 4AH. Thus, if the base of your Dynatrace is 0400H and you have a subroutine which you wish to incorporate at address 0324H and thereby define the A command, you need only store 24H at address 044AH and 03 at address 044BH. Once the new command has been defined, the user need only type the capital letter corresponding to the command he has just defined to call his subroutine from dynatrace. Any subroutine the user may already have in ROM may be incorporated as a command by the above method, or if used only infrequently it may be called and executed with the C command. If the user so desires, he may expand the size of Dynatrace by entering subroutines beginning at address Base + 0C00 hex. In its present form it takes up about 3K of memory. As you can see, the source is so heavily documented that it is close to 54K.

Charlie Foster, The Brain-Drain Co., 7962 Center Parkway, Sacramento, CA 95823

Commands

Commands are given to Dynatrace by typing a single capital letter followed by amplifying data as described below. All addresses and values are given in hex and all hyphens are issued by Dynatrace as prompting characters. Carriage returns for indicating the end of an entry are not required if the value being entered is of the length expected. Thus, typing 4 hex digits for an address value of 2 hex digits for a byte value will complete the entry and not require a terminating character. Any value being entered with fewer digits than required will, however, require a terminating character such as a carriage return.

B	Toggle binary display of scratch registers and accumulator
C-xxxx	Call the user subroutine at address xxxx. The displayed register data is loaded into the 8080 hardware registers and a normal subroutine call is made to the above address. A return to Dynatrace is effected by maintaining proper stack discipline and executing a normal subroutine return instruction. Upon return, the contents of the hardware registers are stored in the user's registers and displayed on the screen of the monitor. The C command must have the user's stack pointer defined to be in some area of existing RAM not used by Dyna-

trace. This is set up initially for the user, so no problems should occur. However, if the user inadvertantly changes the stack pointer to some area in the address space where RAM does not exist, or where it interferes with a stored program (either under test or with Dynatrace itself), results will be unpredictable.

D
FROM-xxxx
TO-yyyy

Display the contents of memory from address xxxx to yyyy. The D command does not like addresses with unlike signs.

G-xxxx
STOP AT-yyyy

Causes simulated execution of a user's program to begin at address xxxx and continue until address yyyy is encountered or any key is typed on the keyboard. The speed of simulation is selected with the "I" command below.

Ix

Sets instruction execution speed from 0 to F hex where 0 is approximately 1 instruction per second and F is about 200 instructions per second.

Lrxxxx

Load register pair with xxxx where "r" is the first letter of the register pair name. (LH1234 put 1234 in the HL register pair.) Note: The accumulator and the Processor Status Word are concatenated to form a register pair "APSW."

Mxxxx-yy-

Store yy at memory location xxxx and prompt with a hyphen for more data. Each succeeding byte will be placed in a subsequent memory location. The entry of data continues until a carriage return is entered in place of information. The M command has no provision for backspacing to delete an erroneously entered character. In the event that an entry is made incorrectly, it will be necessary to terminate the current command line and begin entering data after the last correct entry. The data entered before the error entry will have been stored correctly in memory and hence need not be repeated.

R

Read Intel format papertape from teletype or reader.

S

Causes the simulated execution of one instruction at the location in memory indicated by the user's program counter.

Z
FROM-xxxx
TO-yyyy

Zero RAM from address xxxx up to but not including yyyy.

Peculiarities

1. Due to the nature of the execution of certain instructions, the ending of the execution of one instruction and beginning of the execution of the subsequent instruction does not always correspond with the display. The simulation, however, is always carried out correctly and causes no problems in the program under development. The inconsistency is the updating of the program counter which is delayed one instruction for jumps, calls and returns.
2. Dynatrace is not ROMABLE as is, but if the user has a need for Dynatrace in ROM he can contact the authors to make arrangements to customize Dynatrace to his system. If there is any other need for customizing, the authors are willing to discuss the problem.
3. Typing C-xxxx, where xxxx is the base address of Dynatrace, can be used to restart Dynatrace and clear the screen of any previously entered data. The contents of memory are not disturbed, providing a convenient way to clear the user's registers and reset the stack pointer to its default value (C-8000 in this version).
4. If the user so desires, he may expand the size of Dynatrace by entering subroutines beginning at address Base + 0C00 hex.
5. Your system must be memory-mapped.
6. All commands must be in capitals. An "Escape" will abort an entry.
7. Displays will be at top of screen for registers. The lower middle is for memory read out and the bottom for command lines.

Further Notes

I would like to say something about how to get this program up and running. First of all, it must be edited for the change in EQUATEs that will allow it to run on the user's system. (In my case, I only needed to change the Video and Keyboard equates.) Then it must be assembled. Now, the user only needs to use DDT to call up DYNATRACE.HEX. Once there, type G8000 and DDT will jump into DYNATRACE. To return to DDT send DYNATRACE to a memory location with a RST 7. (If you don't know where one is, use Dynatrace to write one into memory. Then use the C command to go there.)

When you want to debug a program you only need to use DDTs' "I" command to call it up and the "R" command to read it into memory. From there you are on your own. In my system I have a 4K monitor residing in EPROM, so not only do I use Dynatrace—I use my built-in monitor subroutines, too.

If you prefer a COM file, use a Relocatable assembler such as Microsofts' M80 or Cromemcos' ASMB. They can place Dynatrace at any location that you would want. A COM file would have to be placed at 100H.

Conclusion

Finally, as you can see, the program is a long one to type. So for those who would prefer to have the source already on a disk, the authors can provide a copying service for a limited period of time (until Jan. 1, 1982). If the reader will send a self-addressed, stamped shipping package with a disk, the authors will copy and mail their package (by return mail) for a handling fee of $5.00. For those who don't want to bother with any of that, just send $25.00 and the authors will provide everything. At least until the price of materials goes up. Note: The disk will be CPM/soft-sectored/single density.

The authors can be reached at:

THE BRAIN-DRAIN CO.
7962 Center Pkwy
Sacramento, CA 95823

THE DYNATRACE SOURCE

DYNATRACE
V2.0

Edited for publication by Charlie Foster, 1980.

```
        ORG     0500CH
DYNAT:  DS      0       ;BASE ADDRESS OF PROGRAM
STACK   EQU     DYNAT+0CC0H     ;SET UP SYSTEM STACK
CONPRT  EQU     0C8H    ;CONTROL PORT ADDRESS
SCREEN  EQU     0CC00H  ;1K OF BUFFER FOR SCREEN
TOS     EQU     (SCREEN+1024)/256       ;LAST BUFFER ADDRESS+1
MIDSCR  EQU     SCREEN+448      ;TOP LINE OF ROLLUP PORTION OF SCREEN
LINK    EQU     64      ;64 CHARACTERS/LINE
LINE1   EQU     SCREEN+LINK
LINE2   EQU     LINE1+64
LINE3   EQU     LINE2+64
LINE4   EQU     LINE3+64
LINE5   EQU     LINE4+64
LINE15  EQU     SCREEN+960
PCH     EQU     LINE1+8 ;PC HEX DISPLAY LOCATION
INSTA   EQU     LINE2+8 ;INSTRUCTION ASCII DISPLAY LOCATION
INSTH   EQU     LINE2+1B        ;INSTRUCTION HEX DISPLAY LOCATION
SB      EQU     LINE4+8 ;SIGN FLAG BINARY DISPLAY LOCATION
ZB      EQU     LINE4+9 ;ZERO FLAG BINARY DISPLAY LOCATION
ACB     EQU     LINE4+11        ;AUX CARRY FLAG BINARY DISPLAY LOCATION
PB      EQU     LINE4+13        ;PARITY FLAG BINARY DISPLAY LOCATION
CYB     EQU     LINE4+15        ;CARRY FLAG BINARY DISPLAY LOCATION
ACCH    EQU     LINE1+22        ;ACCUMULATOR HEX DISPLAY LOCATION
ACCB    EQU     LINE1+25        ;ACCUMULATOR BINARY DISPLAY LOCATION
BCH     EQU     LINE1+37        ;B REG HEX DISPLAY LOCATION
BCB     EQU     LINE1+42        ;B REG BINARY DISPLAY LOCATION
DEH     EQU     LINE2+37        ;DE REG PR HEX DISPLAY LOCATION
DEB     EQU     LINE2+42        ;DE REG PR BINARY DISPLAY LOCATION
HLH     EQU     LINE3+37        ;HL REG PR HEX DISPLAY LOCATION
HLB     EQU     LINE3+42        ;HL REG PR BINARY DISPLAY LOCATION
SPH     EQU     LINE4+37        ;SP REG PR HEX DISPLAY LOCATION
KSTAT   EQU     6BH
KEYBRD  EQU     6CH
KBDRDY  EQU     80H
RSTAT   EQU     07H
READER  EQU     06B
RDRRDY  EQU     02H
SWTCHS  EQU     0FFH
CR      EQU     0DH
LF      EQU     0AH
ESC     EQU     1BH
;
;
;STORAGE DEFINITION STATEMENTS
;
        ORG     DYNAT+0B80H     ;SYSTEM WORKING STORAGE AREA
PC:     DS      2       ;USER'S PROGRAM COUNTER STORAGE
STKPTR: DS      2       ;USER'S STACK POINTER STORAGE
BC:     DS      2       ;USER'S BC REG PR STORAGE
DE:     DS      2       ;USER'S DE REG PR STORAGE
HL:     DS      2       ;USER'S HL REG PR STORAGE
STSREG: DS      1       ;USER'S STATUS FLAG STORAGE
AC:     DS      1       ;USER'S ACCUMULATOR STORAGE
PC:     DS      2       ;USER'S PRIMARY PROGRAM COUNTER STORAGE
BINFLG: DB      0       ;BINARY DISPLAY SWITCH(0=>NO BINARY DISPLAY)
CPOSIT: DS      2       ;CURSER POSITION FOR ROLLUP PORTION OF SCREEN
STKTMP: DS      2       ;TEMPORARY STORAGE FOR SYSTEM STACK
BASE:   DS      2       ;BASE ADDRESS STORAGE FOR VARIOUS ROUTINES
LAST:   DS      2       ;LAST
FROM:   DB      'FROM-' ;MESSAGES
TO:     DB      'TO-'   ;
ASCII:  DB      '0123456789ABCDEF'      ;ASCII HEX DIGIT TABLE
MOVENM: DB      'MOV , '        ;MOVE MNEMONIC
        DB      'COPYRIGHT (C) 1976, RICHARD E. MEADOR'
;
;CODE BEGINS HERE
;
        ORG     DYNAT   ;START ADDRESS
START:  LXI     SP,STACK        ;DEFINE SYSTEM STACK
        CALL    CLRSCR  ;CLEAR VIDEO SCREEN
        CALL    SETSCR  ;SET UP DISPLAY OF USER REGISTERS
        MVI     A,0     ;CLEAR ACCUMULATOR
        MVI     B,14    ;SET CLEAR COUNT
        LXI     H,PC:   ;SET FIRST ADDRESS TO BE CLEARED
VDM010: MOV     M,A     ;CLEAR
        INX     H       ;NEXT ADDRESS
        DCR     B       ;1 LESS TO DO
        JNZ     VDM010  ;DONE?
        LXI     H,PC:           ;YES, DEFINE USER'S STACK POINTER
        SHLD    STKPTR      ;
VDM015: CALL    DSREGS  ;DISPLAY CONTENTS OF USER'S REGISTERS
        CALL    KEYBDI  ;GET COMMAND
        CPI     40H     ;CHECK FOR ALPHA
        JP      VDM020  ;IGNORE IF NOT
        CPI     5BH     ;
        JP      VDM030  ;
        LXI     H,KEYTAB        ;GET COMMAND LOCATOR TABLE BASE ADDRESS
        SBI     40H     ;SUBTRACT ASCII BIAS FROM RECEIVED COMMAND
        RLC             ;DOUBLE FOR WORD INDEXING
        ADD     L       ;ADD INDEX
        MOV     L,A     ;
        MVI     A,0     ;
        ADC     H       ;
        MOV     H,A     ;
        MOV     E,M     ;GET COMMAND ADDRESS FROM TABLE
        INX     H       ;
        MOV     D,M     ;
        LXI     B,VDM020        ;SET UP RETURN ADDRESS
        PUSH    B       ;SAVE ON STACK
        XCHG            ;HL=COMMAND ROUTINE ADDRESS
```

```
        PCHL            ;(THIS IS REALLY A CALL TO A COMMAND ROUTINE)
VDM020: LXI     H,LINE15        ;RESET CURSOR
        SHLD    CPOSIT  ;
        JMP     VDMG15  ;END OF COMMAND PROCESSING LOOP
;
;COMMAND LOCATION TABLE
;
KEYTAB: DW      UTURN   ;A-NOP
        DW      BINARY  ;B-BINARY DISPLAY TOGGLE
        DW      CALSUB  ;C-CALL USER SUB-PROGRAM
        DW      DUMPME  ;D-DUMP MEMORY TO SCREEN
        DW      UTURN   ;E-NOP
        DW      UTURN   ;F-NOP
        DW      GO      ;G-EXECUTE USER PROGRAM INTERPRETIVELY
        DW      UTURN   ;H-NOP
        DW      ISPEED  ;I-SET EXECUTION SPEED OF INTERPRETER
        DW      UTURN   ;J-NOP
        DW      UTURN   ;K-NOP
        DW      LOADRG  ;L-LOAD REGISTER PAIR
        DW      MEMSTR  ;M-STORE BYTES IN CONSECUTIVE MEMORY LOCATIONS
        DW      UTURN   ;N-NOP
        DW      UTURN   ;O-NOP
        DW      UTURN   ;P-NOP
        DW      UTURN   ;Q-NOP
        DW      READTP  ;R-READ PAPER TAPE
        DW      STEP    ;S-INTERPRET ONE INSTRUCTION
        DW      UTURN   ;T-NOP
        DW      UTURN   ;U-NOP
        DW      UTURN   ;V-NOP
        DW      UTURN   ;W-NOP
        DW      UTURN   ;X-NOP
        DW      UTURN   ;Y-NOP
        DW      ZERMEM  ;Z-ZERO MEMORY
;
;
;
UTURN:  RET             ;DUMMY ROUTINE FOR NOP'S
;
;
;  CLEAR VDM ROUTINE
;
CLRSCR: MVI     A,0     ;CONTROL PORT INITIALIZATION WORD
        OUT     CONPRT  ;SEND
        LXI     B,SCREEN        ;BASE ADDRESS OF VDM BUFFER
CLR010: MVI     A,' '   ;ASCII SPACE
        STAX    B       ;CLEAR 1 BUFFER WORD
        INX     B       ;UPDATE BUFFER POINTER
        MVI     A,7FH AND 0FFH   ;LAST BUFFER ADDRESS +1
        CMP     B       ;CHECK FOR END OF BUFFER AREA
        JNZ     CLR010  ;DO UNTIL DONE
        LXI     H,LINE15        ;INITIAL CURSOR LOCATION
        SHLD    CPOSIT  ;STORE TO CURSOR SAVE
        RET             ;EXIT
;
;  SET DISPLAY OF USER'S REGISTERS
;
SETSCR: LXI     H,'PC'          ;PC
        SHLD    LINE1+5
        MVI     A,'A'           ;A
        STA     LINE1+20
        LXI     H,'BC'          ;BC
        SHLD    LINE1+34
        LXI     H,'IN'          ;IN
        SHLD    LINE2+2
        LXI     H,'ST'          ;ST
        SHLD    LINE2+4
        MVI     A,'R'           ;R
        STA     LINE2+6
        LXI     H,'DE'          ;DE
        SHLD    LINE2+34
        MVI     A,'C'           ;C
        STA     LINE3+15
        MVI     A,'Z'           ;Z
        STA     LINE3+9
        MVI     A,'P'           ;P
        STA     LINE3+13
        MVI     A,'S'           ;S
        STA     LINE3+8
        MVI     A,'A'           ;A
        STA     LINE3+11
        LXI     H,'HL'          ;HL
        SHLD    LINE3+34
        LXI     B,'SP'          ;SP
        SHLD    LINE4+34
        LXI     H,'PS'
        SHLD    LINE4+4
        MVI     A,'N'
        STA     LINE4+6
        MVI     A,'O'
        STA     LINE4+10
        STA     LINE4+12
        INR     A
        STA     LINE4+14
        RET
;
;MOVE THE NUMBER OF BYTES INDICATED IN DE
;FROM THE ADDRESS BEGINNING IN BC
;TO THE ADDRESS BEGINNING IN HL
;
SETLNE: LDAX    B       ;GET BYTE
        MOV     M,A     ;TRANSFER
        INX     H       ;UPDATE DESTINATION POINTER
        INX     B       ;UPDATE SOURCE POINTER
        DCX     D       ;UPDATE COUNT
        MOV     A,E     ;CHECK FOR COMPLETION
        CPI     0       ;
        JNZ     SETLNE  ;
        MOV     A,D     ;
        CPI     0       ;
        JNZ     SETLNE  ;
        RET             ;EXIT
;
; DISPLAY REGISTERS ROUTINE
;
;THIS ROUTINE DECODES THE USER'S REGISTER INFORMATION
;     AND FORMATS IT FOR DISPLAY ON THE CRT SCREEN
;
DSREGS: PUSH    B       ;SAVE ALL REGISTERS
        PUSH    D       ;
        PUSH    H       ;
        PUSH    PSW     ;
        LXI     D,PC1   ;
        LXI     H,PCH+2 ;
        CALL    DSPHEX  ;
        LXI     D,PC1+1 ;
        LXI     H,PCH   ;
        CALL    DSPHEX  ;
        LXI     D,STKPTR        ;
        LXI     H,SPH+2 ;
        CALL    DSPHEX  ;
        LXI     D,STKPTR+1      ;
        LXI     H,SPH   ;
        CALL    DSPHEX  ;
        LXI     D,BC    ;
        LXI     H,BCH+2 ;
        CALL    DSPHEX  ;
        LXI     D,BC+1  ;
        LXI     H,BCH   ;
        CALL    DSPHEX  ;
        LXI     D,DE    ;
        LXI     H,DEH+2 ;
        CALL    DSPHEX  ;
        LXI     D,DE+1  ;
        LXI     H,DEH   ;
        CALL    DSPHEX  ;
        LXI     D,HL    ;
        LXI     H,HLH+2 ;
        CALL    DSPHEX  ;
        LXI     D,HL+1  ;
        LXI     H,HLH   ;
        CALL    DSPHEX  ;
        LXI     D,AC    ;
        LXI     H,ACCH  ;
        CALL    DSPHEX  ;
        LDA     STSWRD  ;
        LXI     H,CYB   ;
        RAR             ;
        MVI     M,'0'   ;
        JNC     DSR010  ;
        MVI     M,'1'   ;
DSR010: RAR             ;
        RAR             ;
        LXI     H,PS    ;
        MVI     M,'0'   ;
        JNC     DSR020  ;
        MVI     M,'1'   ;
DSR020: RAR             ;
        RAR             ;
        LXI     H,ACB   ;
        MVI     M,'0'   ;
        JNC     DSR030  ;
        MVI     M,'1'   ;
DSR030: RAR             ;
        RAR             ;
        LXI     H,EB    ;
        MVI     M,'0'   ;
        JNC     DSR040  ;
        MVI     M,'1'   ;
DSR040: RAR             ;
        LXI     H,SB    ;
        MVI     M,'0'   ;
        JNC     DSR050  ;
        MVI     M,'1'   ;
DSR050: POP     PSW     ;
        POP     H       ;
        POP     D       ;
        POP     B       ;
        LDA     BINFLG  ;CHECK FOR BINARY DISPLAY FLAG SET
        ANA     A       ;
        RZ              ;DONE IF NOT SET
        PUSH    B       ;
        PUSH    D       ;
        PUSH    H       ;
        PUSH    PSW     ;
        LXI     D,AC    ;
        LXI     H,ACCB  ;
        CALL    DSPBIN  ;
        LXI     D,BC    ;
        LXI     H,BCB+9 ;
        CALL    DSPBIN  ;
        LXI     D,BC+1  ;
        LXI     H,BCB   ;
        CALL    DSPBIN  ;
        LXI     D,DE    ;
        LXI     H,DEB+9 ;
        CALL    DSPBIN  ;
        LXI     D,DE+1  ;
        LXI     H,DEB   ;
        CALL    DSPBIN  ;
        LXI     D,HL    ;
        LXI     H,HLB+9 ;
        CALL    DSPBIN  ;
        LXI     D,HL+1  ;
        LXI     H,HLB   ;
        CALL    DSPBIN  ;
        POP     PSW     ;
        POP     H       ;
        POP     D       ;
        POP     B       ;
        RET             ;
;
; DECODE REGISTERS FOR DISPLAY ROUTINE
;
DSPHEX: LDAX    D       ;GET REGISTER CONTENTS
        RRC             ;SCALE ACCUMULATOR DOWN TO GET UPPER HEX DIGIT
        RRC             ;
        RRC             ;
        RRC             ;
        ANI     0FH     ;MASK OFF UNWANTED BITS
        LXI     B,ASCII ;BASE ADDRESS OF ASCII DIGIT TABLE
        ADD     C       ;ADD ACCUMULATOR TO BC TO GET ADDRESS OF DIGIT
        MOV     C,A     ;
        MVI     A,0     ;
        ADC     B       ;
        MOV     B,A     ;
        LDAX    B       ;
        MOV     M,A     ;STORE ASCII CODE FOR HEX DIGIT
        INX     H       ;INCREMENT SCREEN POSITION
        LDAX    D       ;GET REGISTER CONTENTS AGAIN
        ANI     0FH     ;DO THE SAME FOR THE LOWER HEX DIGIT
        LXI     B,ASCII ;
        ADD     C       ;
        MOV     C,A     ;
        MVI     A,0     ;
        ADC     B       ;
```

```
        MOV     B,A     ;
        LDAX    B       ;
        MOV     M,A     ;
        RET             ;
;
; DECODE REGISTER TO BINARY DISPLAY ROUTINE
;
DSPBIN: LDAX    D       ;GET USER REGISTER CONTENTS
        MVI     C,8     ;# OF BITS
DSP010: RAL             ;SHIFT UPPER BIT TO CARRY FOR CHECKING
        MVI     M,'0'   ;ASSUME ZERO
        JNC     DSP020  ;CHECK FOR ONE
        MVI     M,'1'   ;CHANGE IF ONE
DSP020: INX     H       ;MOVE TO NEXT SCREEN POSITION
        DCR     C       ;COUNT OFF
        JNZ     DSP010  ;DONE?
        RET             ;YES
;
;SIMULATOR CODE BEGINS HERE
;  ONE BASIC INSTRUCTION AS INDICATED BY THE USER'S PC REGISTER
STEP:   LHLD    PC      ;GET USER'S PROGRAM COUNTER VALUE
        SHLD    PC1     ;SAVE FOR LAGGING DISPLAY
        MOV     A,M     ;GET THE CONTENTS OF MEMORY LOCATION INDICATED
        CPI     40H     ;CHECK FOR CODES 0-3F HEX
        JC      STP010  ;LOW ORDER 64 OPCODES?
        CPI     80H     ;NO, CHECK FOR 40-7F HEX
        JNC     STP020  ;IS A REGISTER TO REGISTER MOVE INSTRUCTION?
        CALL    MOVE    ;YES, EXECUTE IT
        RET             ;RETURN TO MONITOR
STP010: CALL    OPLOW   ;LOW ORDER OP CODES
        RET             ;RETURN TO MONITOR
STP020: CPI     0C0H    ;CHECK FOR 80-BF HEX
        JNC     STP030  ;ARITHMETIC INSTRUCTION?
        CALL    ARITH   ;YES, EXECUTE
        RET             ;RETURN TO MONITOR
STP030: CALL    OPHIGH  ;HIGH ORDER OP CODE
        RET             ;RETURN TO MONITOR
;
; EXECUTE INSTRUCTION ROUTINE
;
OPEXEC: LXI     H,0     ;SAVE MONITOR'S STACK POINTER
        DAD     SP
        SHLD    STKTMP
        LHLD    STSWRD  ;LOAD REAL REGISTERS WITH USER'S DATA
        PUSH    H
        POP     PSW
        LHLD    STKPTR
        SPHL
        LHLD    BC
        MOV     B,H
        MOV     C,L
        LHLD    DE
        XCHG
        LHLD    HL
INSTR:  DS      3       ;INSTRUCTION TO BE EXECUTED IS STORED HERE
                        ; IF IT WILL NOT CAUSE LOSS OF CONTROL
        SHLD    HL      ;SAVE USER REGISTER VALUES
        PUSH    PSW     ;
        POP     H       ;
        SHLD    STSWRD  ;
        LXI     H,0     ;
        DAD     SP      ;
        SHLD    STKPTR      ;
        XCHG            ;
        SHLD    DE      ;
        LHLD    STKTMP  ;GET MONITOR'S STACK POINTER
        SPHL            ;
        MOV     H,B     ;
        MOV     L,C     ;
        SHLD    BC      ;
        RET             ;
;
; MOV COMMAND EXECUTION ROUTINE
;
MOVE:   MOV     B,A     ;SAVE OP CODE
        ANI     07H     ;DECODE SOURCE REGISTER
        MVI     D,0     ;
        MOV     E,A     ;
        LXI     B,RGSTRS        ;
        DAD     D       ;
        MOV     A,M     ;GET REGISTER NAME
        STA     MOVENM+6    ;SET UP MNEUMONIC
        MOV     A,B     ;RESTORE OP CODE
        RRC             ;
        RRC             ;
        RRC             ;
        ANI     07H     ;DO SAME FOR DESTINATION REGISTER
        MOV     E,A     ;
        LXI     H,RGSTRS        ;
        DAD     D       ;
        MOV     A,M     ;
        STA     MOVENM+4    ;
        MOV     A,B     ;
        MVI     B,1     ;# OF BYTES IN INSTRUCTION
        CALL    MVINST  ;COPY INSTRUCTION INTO EXECUTION AREA
        LXI     B,INSTA ;DISPLAY MNEUMONIC.
        LXI     H,MOVENM    ;
        LXI     D,8     ;
        CALL    SETLNE  ;
        CALL    OPEXEC  ;GO EXECUTE INSTRUCTION
        RET             ;DONE.
;
; ARITHMETIC INSTRUCTION EXECUTION ROUTINE
;
ARITH:  MOV     B,A     ;SAVE OP CODE
        LXI     H,ARITHN        ;GET ADDRESS OF ARITH MNEUMONICS
        ANI     38H     ;ISOLATE REGISTER OPERAND
        MVI     D,0     ;
        MOV     E,A     ;
        DAD     D       ;ADD INDEX TO GET APPROPRIATE MNEUMONIC
        PUSH    H       ;SAVE TEMPORARILY
        MOV     A,B     ;RESTORE OPCODE
        ANI     07H     ;ISOLATE REGISTER OPERAND #
        LXI     H,RGSTRS        ;GET REGISTER NAME LIST ADDRESS
        MOV     E,A     ;
        DAD     D       ;ADD INDEX TO GET CORRECT REGISTER MNEUMONIC
        MOV     A,M     ;GET MNEUMONIC
        POP     H       ;RETRIEVE INSTRUCTION MNEUMONIC ADDRESS
        MVI     E,4     ;
        DAD     D       ;
        MOV     M,A     ;MOVE REGISTER NAME TO MEMORY
        DCX     SP      ;
        DCX     SP      ;
        MOV     A,B     ;RESTORE OP CODE
        MVI     B,1     ;SET 1 OF BYTES IN OPCODE
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        LXI     H,INSTA ;DISPLAY MNEUMONIC
        LXI     D,8     ;
        POP     B       ;
        CALL    SETLNE  ;
        CALL    OPEXEC  ;GO EXECUTE INSTRUCTION
        RET             ;
;
;ARITHMETIC INSTRUCTION MNEUMONICS
;
ARITHN: DB      'ADD    '       ;ADD REGISTER TO ACCUMULATOR
        DB      'ADC    '       ;ADD REGISTER+CARRY TO ACCUMULATOR
        DB      'SUB    '       ;SUB REGISTER FROM ACCUMULATOR
        DB      'SBB    '       ;SUB REGISTER+CARRY FROM ACCUMULATOR
        DB      'ANA    '       ;LOGICAL AND REGISTER WITH ACCUMULATOR
        DB      'XRA    '       ;LOGICAL EXCLUSIVE OR REGISTER WITH ACCU
        DB      'ORA    '       ;LOGICAL OR REGISTER WITH ACCUMULATOR
        DB      'CMP    '       ;COMPARE REGISTER WITH ACCUMULATOR
;
; LOW ORDER INSTRUCTION EXECUTION ROUTINE
; OP CODES 0-3F HEX
OPLOW:  MOV     E,A     ;SAVE OP CODE
        ANI     07H     ;DETERMINE CLASS AND BRANCH TO APPROPRIATE ROUTINE
        RLC             ;
        MOV     C,A     ;
        MVI     B,0     ;
        LXI     H,OPTAB ;
        DAD     B       ;
        MOV     A,M     ;
        MOV     E,M     ;
        INX     H       ;
        MOV     D,M     ;
        XCHG            ;
        PCHL            ;
;
;OP CODE SIMULATION ROUTINE LOOKUP TABLE
;
OPTAB:  DW      NPINST      ; NO-OP ROUTINE POINTER
        DW      LXIDAD  ; LXI & DAD ROUTINE POINTER
        DW      LDSTRX  ; LDAX & STAX ROUTIN POINTER
        DW      INXDCX  ;INX & DCX ROUTINE POINTER
        DW      INRDCR  ; INR & DCR ROUTINE POINTER
        DW      INRDCR  ; SAME
        DW      MVIMM     ; MVI ROUTINE POINTER
        DW      ROTATE  ; ROTATE POINTER
;
; A=OP CODE, B= NO. OF BYTES
;
MVINST: STA     INSTR   ;STORE OP CODE TO EXECUTION AREA
        PUSH    B       ;SAVE NUMBER OF BYTES IN OP CODE
        CALL    HEXASC  ;CONVERT OP CODE TO ASCII/HEX EQUIVALANT
        MOV     H,C     ;TRANSFER FOR DOUBLE LENGTH STORE
        MOV     L,B     ;
        SHLD    INSTH   ;STORE ASCII CODES OF OP CODE TO VDM BUFFER
        POP     B       ;RETRIEVE NUMBER OF BYTES IN INSTRUCTION
        LXI     D,INSTR+1       ;GET ADDRESS OF NEXT LOCATION IN EXECUTI
        MVI     A,0     ;ZERO NEXT 2 BYTES IN EXECUTION AREA
        STAX    D       ;    IN CASE INSTRUCTION<3 BYTES
        STA     INSTR+2 ;
        MVI     H,' '   ;CLEAR PREVIOUS EXECUTION FROM SCREEN
        MOV     L,H     ;
        SHLD    INSTH+2 ;
        SHLD    INSTH+4 ;
        LXI     H,INSTH+2       ;SET TO DISPLAY REST OF INSTRUCTION IF A
        SHLD    STKTMP  ;SAVE VDM BUFFER LOCATION OF ASCII/HEX DISPLAY A
        LHLD    PC      ;GET USER'S PROGRAM COUNTER
        INX     H       ;UPDATE IT
MVI010: DCR     B       ;COUNT OFF NUMBER OF BYTES IN INSTRUCTION
        JZ      MVI020  ;EXIT IF DONE
        MOV     A,M     ;GET NEXT BYTE OF INSTRUCTION FROM USER'S PROGRA
        INX     H       ;UPDATE USER'S PROGRAM COUNTER
        STAX    D       ;STORE BYTE TO EXECUTION AREA
        INX     D       ;UPDATE EXECUTION BUFFER POINTER
        PUSH    B       ;SAVE NUMBER OF BYTES IN INSTRUCTION
        PUSH    D       ;SAVE EXECUTION BUFFER POINTER
        PUSH    H       ;SAVE USER'S PROGRAM COUNTER
        CALL    HEXASC  ;CONVERT NEXT BYTE OF INSTRUCTION TO ASCII/HEX C
        LHLD    STKTMP  ;RETRIEVE VDM BUFFER LOCATIN OF ASCII/HEX DISPLA
        MOV     M,B     ;MOVE ASCII/HEX CODES OF CURRENT INSTRUCTION BYT
        INX     H       ;    TO VDM BUFFER AREA
        MOV     M,C     ;
        INX     H       ;
        SHLD    STKTMP  ;SAVE VDM BUFFER LOCATION AGAIN
        POP     H       ;RETRIEVE USER'S PROGRAM COUNTER
        POP     D       ;RETRIEVE EXECUTION BUFFER POINTER
        POP     B       ;RETRIEVE NUMBER OF BYTES LEFT IN INSTRUCTION
        JMP     MVI010  ;CONTINUE UNTIL ALL BYTES MOVED
MVI020: SHLD    PC      ;STORE UPDATED USER'S PROGRAM COUNTER
        RET             ;EXIT
;
;
; MOVE MNEUMONIC TO BUFFER
;
MVNMNC: LXI     H,INSTA ;GET VDM BUFFER LOCATION OF MNEUMONIC DISPLAY AS
        PUSH    D       ;SAVE D REGISTER
        LXI     D,4     ;SET NUMBER OF BYTES TO TRANSFER
        CALL    SETLNK  ;DO TRANSFER
        POP     B       ;GET OLD D REGISTER
        MVI     D,' '   ;THE FOLLOWING CODE MOVES THE ASCII CHARACTERS O
        MOV     M,D     ;    THE REGISTERS USED BY THE INSTRUCTION TO THE
        INX     B       ;        THE VDM BUFFER AREA
        MOV     M,C     ;
        INX     B       ;
        MOV     M,B     ;
        INX     B       ;
        MOV     M,D     ;
        RET             ;
;
; GET REGISTER PAIR ROUTINE
;
GETRP:  ANI     30H     ;MASK REGISTER PAIR FIELD
        RRC             ;SCALE FOR INDEXING
        RRC             ;
        RRC             ;
        MOV     E,A     ;BUILD INDEX IN DE
        MVI     D,0     ;
        LXI     H,REGPAR        ;GET BASE ADDRESS OF REG PAIR MNEUMONICS
        DAD     D       ;ADD INDEX
        SHLD    GET010+1        ;SAVE POINTER
GET010: LHLD    0       ;0 IS REPLACED BY POINTER FROM ABOVE
```

```
        XCHG            ;POSITION FOR SUB ROUTINE CALL
        CALL    MVMNMC  ;DISPLAY OP CODE
        RET             ;
;
; EXECUTE NO-OP ROUTINE
;
NPINST:    MVI     B,1     ;SET INSTRUCTION LENGTH
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        LXI     D,'  '  ;NO REGISTER DATA
        LXI     H,LOWOP ;ADDRESS OF NOP MNEUMONIC
        CALL    MVMNMC  ;DISPLAY OP CODE
        CALL    OPEXEC  ;EXECUTE INSTRUCTION
        RET             ;
;
;  EXECUTE LXI AND DAD INSTRUCTIONS ROUTINE
;
LXIDAD: MOV     B,A     ;SAVE INSTRUCTION
        ANI     08H     ;ISOLATE LXI/DAD BIT
        MOV     A,B     ;RESTORE INSTRUCTION
        MVI     B,1     ;ASSUME DAD INSTRUCTION
        JNZ     LXI010  ;DAD OR LXI?
        MVI     B,3     ;LXI, CHANGE INSTRUCTION LENGTH TO 3 BYTES
LXI010: PUSH    PSW     ;SAVE INSTRUCTION DESIGNATOR WHICH IS THE ZERO F
        CALL    MVINST  ;DISPLAY OP CODE
        POP     PSW     ;RETRIEVE FLAG
        LXI     B,LOWOP+4       ;ASSUME DAD
        JNZ     LXI030  ;DAD/LXI?
        LXI     B,LOWOP3        ;LXI, CHANGE POINTER
LXI020: CALL    GETRP   ;ISOLATE REG PAIR
        CALL    OPEXEC  ;EXECUTE INSTRUCTION
        RET             ;
;
; LDA STA LDAX STAX SHLD LHLD  EXECUTION ROUTINE
;
LDSTRX: MOV     C,A     ;SAVE INSTRUCTION
        ANI     20H     ;ISOLATE (LDX,STX)/(LHLD,SHLD,LDA,STA) BIT
        MOV     A,C     ;RESTORE INSTRUCTION
        JNZ     LDS020  ;(LDX,STX)/(LHLD,SHLD,LDA,STA)?
        ANI     08H     ;ISOLATE LDX/STX BIT
        MOV     A,C     ;RESTORE INSTRUCTION
        MVI     B,3     ;INSTRUCTION LENGTH IS 3 FOR BOTH
        PUSH    PSW     ;SAVE DESIGNATOR BIT
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        POP     PSW     ;RETRIEVE DESIGNATOR BIT
        LXI     B,LOWOP2        ;ASSUME STAX
        JZ      LDS010  ;STAX/LDAX?
        LXI     B,LOWOP2+4      ;LDAX, CHANGE MNEUMONIC POINTER
LDS010: CALL    GETRP   ;DETERMINE REGISTER PAIR
        JMP     LDS030  ;GO EXECUTE
LDS020: MVI     B,3     ;(LHLD,SHLD,LDA,STA), SET LENGTH TO 3 BYTES
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        MOV     A,C     ;RESTORE INSTRUCTION
        ANI     18H     ;ISOLATE INSTRUCTION DESIGNATOR
        RRC             ;SCALE FOR INDEXING
        MOV     E,A     ;BUILD INDEX
        MVI     D,0     ;
        LXI     H,LOWOP2+8      ;GET BASE ADDRESS OF INSTRUCTION GROUP
        DAD     D       ;ADD INDEX
        MOV     B,H     ;POSITION FOR CALL
        MOV     C,L     ;
        LXI     D,'  '  ;SET REG TYPE TO NULL
        CALL    MVMNMC  ;DISPLAY CODE
LDS030: CALL    OPEXEC  ;EXECUTE INSTRUCTION
        RET             ;
;
; INX DCX EXECUTION ROUTINE
;
INXDCX: MOV     C,A     ;SAVE INSTRUCTION
        MVI     B,1     ;THESE ARE ALL ONE BYTE
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        MOV     A,C     ;RESTORE INSTRUCTION
        PUSH    PSW     ;SAVE IT AGAIN
        ANI     08H     ;ISOLATE DESIGNATOR BIT
        LXI     B,LOWOP3        ;ASSUME INX
        JZ      INX010  ;INX/DCX?
        LXI     B,LOWOP3+4      ;DCX, CHANGE POINTER
INX010: POP     PSW     ;GET INSTRUCTION BACK
        CALL    GETRP   ;DETERMINE REG PAIR
        CALL    OPEXEC  ;EXECUTE INSTRUCTION
        RET             ;
;
; INR DCR EXECUTION ROUTINE
;
INRDCR: MOV     C,A     ;SAVE INSTRUCTION
        MVI     B,1     ;THESE ARE ALL ONE BYTE
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        MOV     A,C     ;RESTORE INSTRUCTION
        STA     STKTMP  ;SAVE IT AGAIN
        LXI     B,LOWOP4        ;ASSUME INR
        ANI     01H     ;CHECK DESIGNATOR BIT
        JZ      INR010  ;INR/DCR?
        LXI     B,LOWOP5        ;DCR, CHANGE POINTER
INR010: LDA     STKTMP  ;GET INSTRUCTION BACK
        CALL    GETREG  ;DETERMINE REGISTER
        CALL    OPEXEC  ;EXECUTE INSTRUCTION
        RET             ;
;
; MVI EXECUTION ROUTINE
;
MVIMM:     MOV     C,A     ;SAVE INSTRUCTION
        MVI     B,2     ;THESE ARE ALL 2 BYTE INSTRUCTIONS
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        MOV     A,C     ;RESTORE INSTRUCTION
        LXI     B,LOWOP6        ;SET POINTER MVI MNEUMONIC
        CALL    GETREG  ;DETERMINE REGISTER
        CALL    OPEXEC  ;EXECUTE INSTRUCTION
        RET             ;
;
; RLC RRC RAL RAR DAA CMA STC CMC
; EXECUTION ROUTINE
;
ROTATE: MOV     C,A     ;SAVE INSTRUCTION
        MVI     B,1     ;THESE ARE ALL ONE BYTE INSTRUCTIONS
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        MOV     A,C     ;RESTORE INSTRUCTION
        ANI     38H     ;ISOLATE DESIGNATOR BITS
        RRC             ;SCALE FOR INDEXING
        MOV     E,A     ;CONSTRUCT INDEX
        MVI     D,0     ;
        LXI     H,LOWOP7        ;GET BASE OF ADDRESS OF MNEUMONICS
        DAD     D       ;ADD INDEX
        MOV     C,L     ;POSITION ADDRESS FOR SUBROUTINE CALL
        MOV     B,H     ;
        LXI     D,'  '  ;SET REGISTER TO NULL
        CALL    MVMNMC  ;DISPLAY OP CODE
        CALL    OPEXEC  ;EXECUTE INSTRUCTION
        RET             ;
;
; GET REGISTER ROUTINE
;
GETREG: ANI     38H     ;ISOLATE REGISTER DESIGNATOR BITS
        RRC             ;SCALE FOR INDEXING
        RRC             ;
        RRC             ;
        MOV     E,A     ;CONSTRUCT INDEX WORD
        MVI     D,0     ;
        LXI     H,RGSTRS        ;GET BASE ADDRESS OF REGISTER CODES
        DAD     D       ;ADD INDEX
        MVI     E,' '   ;OTHER REGISTER IS NULL
        MOV     D,M     ;GET THIS REGISTER
        CALL    MVMNMC  ;DISPLAY OP CODE AND REGISTER
        RET             ;
LOWOP:  DB      'NOP '  ;NO-OP
LOWOP1: DB      'LXI '  ;LOAD REGISTER PAIR IMMEDIATE(EXTENDED)
        DB      'DAD '  ;DOUBLE LENGTH ADD
LOWOP2: DB      'STAX'  ;STORE ACCUMULATOR INDIRECT THROUGH REG PAIR
        DB      'LDAX'  ;LOAD ACCUMULATOR INDIRECT THROUGH REG PAIR
        DB      'SHLD'  ;STORE HL DIRECT
        DB      'LHLD'  ;LOAD HL DIRECT
        DB      'STA '  ;STORE ACCUMULATOR DIRECT
        DB      'LDA '  ;LOAD ACCUMULATOR DIRECT
LOWOP3: DB      'INX '  ;INCREMENT REGISTER PAIR
        DB      'DCX '  ;DECREMENT REGISTER PAIR
LOWOP4: DB      'INR '  ;INCREMENT REGISTER
LOWOP5: DB      'DCR '  ;DECREMENT REGISTER
LOWOP6: DB      'MVI '  ;MOVE DATA IMMEDIATE TO REGISTER
LOWOP7: DB      'RLC '  ;ROTATE LEFT THROUGH CARRY
        DB      'RRC '  ;ROTATE RIGHT THROUGH CARRY
        DB      'RAL '  ;ROTATE ARITHMETIC LEFT
        DB      'RAR '  ;ROTATE ARITHMETIC RIGHT
        DB      'DAA '  ;DECIMAL ADJUST ACCUMULATOR
        DB      'CMA '  ;COMPLIMENT ACCUMULATOR
        DB      'STC '  ;SET CARRY
        DB      'CMC '  ;COMPLIMENT CARRY
RGSTRS: DB      'BCDEHLMA'      ;REGISTER CODES
REGPAR: DB      'B D H SP'      ;REGISTER PAIR CODES
FLAGS:  DB      'NZZ NCC POPEP M '      ;STATUS FLAG CODES
;
;
;
KEYBDI: PUSH    H       ;SAVE HL
        LHLD    CPOSIT  ;GET CURSOR POSITION
KEY005: IN      KSTAT   ;CHECK 3P+S STATUS
        ANI     KBDRDY  ;LOOK FOR KEY BOARD READY
        JNZ     KEY005  ;LOOP ON NO DATA AVAILABLE
        IN      KEYBRD  ;GET DATA FROM ASCII KEYBOARD
        ANI     7FH     ;STRIP OFF PARITY BIT
        MOV     B,A     ;SAVE CHARACTER
        CPI     ESC     ;CHECK FOR ESCAPE SEQUENCE
        JNZ     KEY007  ;WAS IT?
        LXI     SP,STACK        ;YES, RESET STACK POINTER
        CALL    BMPLNE  ;SCROLL UP ONE LINE
        JMP     VDM015  ;RESUME SCAN LOOP
KEY007: CPI     CR      ;CHECK FOR CARRIAGE RETURN
        JZ      KEY020  ;WAS IT
KEY010: MOV     M,A     ;NO, DISPLAY CHARACTER
        INX     H       ;UPDATE CURSER
        MOV     A,H     ;CHECK FOR VDM BUFFER OVERFLOW
        CPI     7DS AND 0FFH    ;
        JC      KEY030  ;
KEY020: CALL    BMPLNE  ;SCROLL UP ON BUFFER FULL
        LXI     H,LINE15        ;RESET CURSOR TO BEGINNING OF LAST LINE
KEY030: MOV     A,B     ;RESTORE CHARATER RECEIVED FROM KEYBOARD
        SHLD    CPOSIT  ;STORE UPDATED CURSOR
        POP     H       ;RESTORE ORIGINAL CONTENTS OF HL
        RET             ;
;
;
;
BMPLNE: PUSH    B       ;SAVE HL
        PUSH    D       ;SAVE DE
        PUSH    B       ;SAVE BC
        PUSH    PSW     ;SAVE PSW
        LXI     H,MIDSCR        ;BUFFER ADDRESS TO MOVE OLD DATA TO
        LXI     B,MIDSCR+LINE   ;BUFFER ADDRESS TO GET OLD DATA FROM
        MVI     E,8     ;NUMBER OF LINES TO BE SCROLLED
BMP010: PUSH    D       ;SAVE
        LXI     D,64    ;NUMBER OF CHARACTERS PER LINE
        CALL    SETLNE  ;SCROLL ONE LINE AT A TIME FROM THE TOP DOWN
        POP     D       ;GET NUMBER OF LINES LEFT TO GO
        DCR     E       ;COUNT OFF ONE MORE
        JNZ     BMP010  ;DO UNTIL DONE
        MVI     A,' '   ;STE TO SPACE OUT LAST LINE
        MVI     D,64    ;BLANK OUT ALL CHARACTERS ON LAST LINE
        LXI     H,LINE15        ;GET ADDRESS OF FIRST CHARACTER OF LAST
BMP020: MOV     M,A     ;BLANK IT OUT
        INX     H       ;UPDATE POINTER
        DCR     D       ;COUNT DOWN
        JNZ     BMP020  ;DO WHILE NOT DONE
        LXI     H,LINE15        ;RESET CURSOR
        SHLD    CPOSIT  ;
        POP     PSW     ;RESTORE ORIGINAL REGISTER CONTENTS
        POP     B       ;
        POP     D       ;
        POP     H       ;
        RET             ;
;
;
;
BINARY: LDA     BINFLG  ;GET BINARY DISPLAY FLAG
        CMA             ;TOGGLE IT
        STA     BINFLG  ;PUT IT BACK
        CALL    CLRSCR  ;RESET SCREEN
        CALL    SETSCR  ;
        RET             ;
;
;
;
GTLNTS: CALL    BMPLNE  ;SCROLL UP
        LXI     B,PROM  ;DISPLAY PROM MESSAGE
        LXI     H,LINE15        ;
        LXI     D,5     ;
        CALL    SETLNE  ;
        LXI     H,LINE15+5      ;
        SHLD    CPOSIT  ;
        CALL    GETADR  ;GET ONE HEX NUMBER OF UP TO 4 DIGITS
```

```
SHLD    BASE    ;THAT WAS THE BASE ADDRESS
CALL    BMPLNE  ;SCROLL UP
        LXI     B,TO    ;DISPLAY TO MESSAGE
        LXI     H,LINE15        ;
        LXI     D,3     ;
        CALL    SETLNE  ;
        LXI     H,LINE15+3      ;
        SHLD    CPOSIT  ;
        CALL    GETADR  ;GET ONE HEX NUMBER
        XCHG            ;MOVE TO DE
        LHLD    BASE    ;GET BASE ADDRESS
        CALL    BMPLNE  ;SCROLL UP
        RET             ;
;
;
;
DUMPME: CALL    GTLM75  ;GET DUMP LIMITS
DMPOI0: PUSH    D       ;SAVE LAST ADDRESS
        CALL    DMEM16  ;DISPLAY 16 BYTES
        POP     D       ;GET LAST ADDRESS
        MOV     A,E     ;SEE IF LAST ADDRESS EXCEEDED
        SUB     L       ;
        MOV     A,D     ;
        SBB     H       ;
        JP      DMPOI0  ;CONTINUE DUMPING IF NOT
        RET             ;
;
;
;
GETBYT: MVI     E,2     ;GET TWO CHARACTERS FROM KEY BOARD
        JMP     GTAC05  ;GO AROUND NEXT ENTRY POINT
GETADR: MVI     E,4     ;GET 4 CHARACTERSFROM KEYBOARD
GTA005: LXI     H,0     ;START WITH ZERO
GTA010: CALL    KEYBDI  ;GET ACBARACTER FROM THE KEYBOARD
        CPI     CR      ;CHECK FOR TERMINATOR
        JZ      GTA020  ;EXIT ON END OF NUMBER
        CALL    ASCHEX  ;CONVERT CHARACTER TO HEX DIGIT
        MOV     C,A     ;POISITION FOR ADD
        MVI     B,0     ;
        DAD     H       ;MULTIPLY HL BY 16
        DAD     H       ;
        DAD     H       ;
        DAD     H       ;
        DAD     B       ;ADD LAST CHARACTER RECIEVED
        DCR     E       ;COUNT OFF NUMBER OF DIGITS
        JNZ     GTA010  ;DO UNTIL COUNT EXAUSTED OR CARRIAGE RETURN
GTA020: RET             ;
;
;
ASCHEX: SBI     '0'     ;SUBTRACT OFF ASCII BIAS
        JM      ASC010  ;CHAR<07
        CPI     10      ;NO,
        RC              ;DONE IF CHAR<=9
        SBI     07H     ;ADJUST FOR ALPHA BIAS
        CPI     LF      ;
        JC      ASC010  ;
        CPI     106     ;CHAR>0FH?
        RC              ;NO, RETURN
ASC010: MVI     A,0     ;RETURN ZERO IF ILLEAGLE
        RET             ;
;
;DUMP 16 BYTES TO TTY
;
DMEM16: MOV     A,L     ;DISPLAY ADDRESS
        CALL    HEXASC  ;
        PUSH    B       ;
        MOV     A,H     ;
        CALL    HEXASC  ;
        LXI     D,LINE15        ;
        XCHG            ;
        MOV     M,B     ;
        INX     H       ;
        MOV     M,C     ;
        INX     H       ;
        POP     B       ;
        MOV     M,B     ;
        INX     H       ;
        MOV     M,C     ;
        MVI     B,16    ;NUMBER OF BYTES TO DISPLAY
DMEU1C: PUSH    B       ;SAVE
        INX     H       ;UPDATE ADDRESS
        LDAX    D       ;GET BYTE FROM MEMORY
        INX     D       ;UPDATE POINTER
        CALL    HEXASC  ;CONVERT TO ASCII CHARACTERS AND DISPLAY
        INX     H       ;
        MOV     M,B     ;
        INX     H       ;
        MOV     M,C     ;
        POP     B       ;
        DCR     B       ;
        JNZ     DMED10  ;DO UNTIL ALL 16 DISPLAYED
        CALL    BMPLNE  ;SCROLL UP
        XCHG            ;
        RET             ;
;
;CONVERT BYTE IN ACCUMULATOR TO ASCII DIGITS
;
HEXASC: PUSH    D       ;SAVE DE
        PUSH    H       ;SAVE HL
        LXI     H,ASCII ;GET BASE ADDRESS OF ASCII CODE
        MOV     B,A     ;SAVE DATA
        ANI     0FH     ;DO LOWER HEX DIGIT FIRST
        MOV     E,A     ;SET UP FOR INDEXING
        MVI     D,0     ;
        DAD     D       ;ADD INDEX
        MOV     C,M     ;GET CHARACTER
        MOV     A,B     ;RESTORE DATA
        ANI     0F0H    ;DO UPPER HEX DIGIT
        RRC             ;SCALE DOWN FOR INDEXING
        RRC             ;
        RRC             ;
        RRC             ;
        MOV     E,A     ;SET UP FOR INDEXING
        LXI     H,ASCII ;GET BASE ADDRESS OF ASCII CODES AGAIN
        DAD     D       ;ADD INDEX
        MOV     B,M     ;GET CHARACTER
        POP     H       ;RESTORE HL
        POP     D       ;RESTORE DE
        RET             ;
```

```
;LOAD REGISTERS FROM KEYBOARD ROUTINE
;
LOADRG: CALL    KEYBDI  ;GET REGISTER IDENTIFIER
        LXI     H,LOREGS        ;DETERMINE SELECTED REGISTER
        MVI     D,6     ;6 POSSIBLE REGISTER PAIRS
        MOV     B,A     ;SAVE REGISTER DESIGNATOR
LOAC10: MOV     A,M     ;GET ONE FROM MEMORY
        CMP     B       ;CHECK AGAINST SELECTED REGISTER PAIR
        JZ      LOA020  ;MATCH
        INX     H       ;NO, KEEP LOOKING
        DCR     D       ;
        JNZ     LOA010  ;
        RET             ;RETURN IF NOT FOUND
LOA020: MOV     A,D     ;GET INDEX NUMBER
        DCR     A       ;ADJUST
        RLC             ;SCALE
        MOV     E,A     ;SET UP INDEX
        MVI     D,0     ;
        PUSH    D       ;SAVE INDEX
        CALL    GETADR  ;GET A HEX NUMBER FROM KEYBOARD
        XCHG            ;RELOCATE NUMBER TO LESS CONVENIENT REGISTER
        POP     H       ;GET INDEX BACK
        MOV     A,L     ;MOVE TO ACCUMULATOR
        ANA     A       ;SET FLAGS
        JNZ     LOA030  ;IS IT THE PC REGISTER?
        PUSH    H       ;YES,
        LXI     B,PC    ;GET ADDRESS OF USER'S PC REG
        DAD     B       ;THIS IS REDUNDANT
        MOV     M,E     ;STORE DATA TO USER'S PC REGISTER
        INX     H       ;
        MOV     M,D     ;
        POP     H       ;
LOA030: LXI     B,PC1   ;STORE DATE TO USER REGISTER
        DAD     B       ;
        MOV     M,E     ;
        INX     H       ;
        MOV     M,D     ;
        CALL    BMPLNE  ;
        RET             ;
LOREGS: DB      'AHDRSP'        ;POSSIBLE REGISTER DESIGNATIONS
;
;STORE DATA TO MEMORY ROUTINE
;
MEMSTR: CALL    GETADR  ;GET BEGINNING ADDRESS
MEM010: PUSH    H       ;SAVE IT
        LHLD    CPOSIT  ;GET CURSOR POSITION
        MVI     A,'-'   ;DISPLAY PROMPT
        MOV     M,A     ;
        INX     H       ;UPDATE CURSOR
        SHLD    CPOSIT  ;RESTORE UPDATED CURSOR
        CALL    GETBYT  ;GET A DATA BYTE
        MOV     A,E     ;GET NUMBER OF BYTES RECIEVED
        CPI     2       ;CHECK FOR NONE RECIEVED
        JNZ     MEM020  ;ANY DATA RECIEVED?
        POP     H       ;NO, RESTORE HL
        RET             ;GET OUT OF HERE
MEM020: MOV     A,L     ;MOVE DATA BYTE TO ACCUMULATOR
        POP     H       ;GET ADDRESS BACK
        MOV     M,A     ;STORE DATA TO MEMORY
        INX     H       ;UPDATE ADDRESS
        JMP     MEM010  ;CONTINUE
;
;CALL USER SUBROUTINE
;
CALSUB: LHLD    CPOSIT  ;GET CURSOR POSITION
        MVI     A,'-'   ;DISPLAY PROMPT
        MOV     M,A     ;
        INX     H       ;UPDATE CURSOR POSITION
        SHLD    CPOSIT  ;RESTORE CURSOR
        MVI     A,0CDH  ;LOAD ACCUMULATOR WITH A CALL INSTRUCTION
        STA     INSTR   ;MOVE TO EXECUTION AREA
        CALL    GETADR  ;GET ADDRESS OF SUBROUTINE TO BE EXECUTED
        SHLD    INSTR+1 ;STORE ADDRESS TO EXECUTION AREA PLUS ONE
        CALL    BMPLNE  ;SCROLL UP
        CALL    OPEXEC  ;EXECUTE SUBROUTINE
        CALL    CLRSCR  ;CLEAR ANY GARBAGE FROM SCREEN
        CALL    SETSCR  ;SET UPSCREEN AGAIN
        RET             ;
;
;HIGH ORDER INSTRUCTION DECODE ROUTINE
;
OPHIGH: MOV     E,A     ;SAVE INSTRUCTION
        ANI     07H     ;DETERMINE SUB CLASS
        RLC             ;SCALE FOR INDEX
        MOV     C,A     ;POSITION FOR INDEXING
        MVI     B,0     ;
        LXI     H,HOPTAB        ;GET HIGH ORDER INSTRUCTION TABLE BASE A
        DAD     B       ;ADD INDEX
        MOV     A,E     ;RESTORE INSTRUCTION
        MOV     E,M     ;GET ADDRESS OF INSTRUCTION ROUTINE
        INX     H       ;
        MOV     D,M     ;
        XCHG            ;MOVE TO HL
        PCHL            ;PASS CONTROL TO APPROPIATE ROUTINE
;
HOPTAB: DW      RETURN  ;RETURN
        DW      PPINST    ;POP
        DW      JUMP    ;JUMP
        DW      MISC    ;MISC
        DW      CLINST    ;CALL
        DW      PSINST    ;PUSH
        DW      IMMED7  ;IMMEDIATE
        DW      RESTRT  ;RESTART
;
MIHOP:  DB      'R   '  ;CONDITIONAL RETURN
SIHOP1: DB      'POP '  ;POP
        DB      'RET '  ;UNCONDITIONAL ROUTINE
        DB      '*NOP'  ;UNIMPLEMENTED OP CODES
        DB      'PCHL'  ;INDIRECT JUMP
        DB      'SPHL'  ;LOAD STACK POINTER WITH HL
MIHOP2: DB      'J   '  ;CONDITIONAL JUMP
MIHOP3: DB      'JMP '  ;UNCONDITIONAL JUMP
        DB      '*NOP'  ;UNIMPLEMENTED OP CODE
        DB      'OUT '  ;OUTPUT
        DB      'IN  '  ;INPUT
        DB      'XTHL'  ;EXCHANGE HL AND TOP OF STACK
        DB      'XCHG'  ;EXCHANGE HL AND DE
        DB      'DI  '  ;DISABLE INTERRUPTS
        DB      'EI  '  ;ENABLE INTERRUPTS
MIHOP4: DB      'C   '  ;CONDITIONAL CALL
MIHOP5: DB      'PUSH'  ;PUSH
        DB      'CALL'  ;UNCONDITIONAL CALL
        DB      '*NOP'  ;UNIMPLEMENTED OP CODE
MIHOP6: DB      'ADI '  ;ADD IMMEDIATE
```

```
        DB      'ACI '  ;ADD IMMEDIATE WITH CARRY
        DB      'SUI '  ;SUBTRACT IMMEDIATE
        DB      'SBI '  ;SUBTRACT IMMEDIATE WITH BORROW
        DB      'ANI '  ;AND IMMEDIATE
        DB      'XRI '  ;EXCLUSIVE OR IMMEDIATE
        DB      'ORI '  ;OR IMMEDIATE
        DB      'CPI '  ;COMPARE IMMEDIATE
#IHOP7: DB      'RST '  ;RESTART
;
;DETERMINE CONDITIONAL FLAG ROUTINE
;
GTFLAG: ANI     38H     ;ISOLATE CONDITION BITS
        PUSH    H       ;SAVE HL
        RRC             ;SCALE FOR INDEXING
        RRC             ;
        MVI     D,0     ;
        MOV     E,A     ;CONSTRUCT INDEX
        LXI     H,FLAGS ;GET BASE ADDRESS OF FLAG CODES
        DAD     D       ;ADD INDEX
        MOV     A,M     ;GET FLAG CODE
        STAX    B       ;STORE TO VDM BUFFER
        INX     B       ;UPDATE POINTER
        INX     H       ;UPDATE FLAG LOCATOR
        MOV     A,M     ;GET SECOND CHARACTER OF FLAG
        STAX    B       ;DISPLAY IT
        LXI     D,'  '  ;OTHER CHARACTERS ARE BLANK
        POP     B       ;GET OLD HL
        CALL    MVMNMC  ;DISPLAY UP CODE
        RET             ;
;
;TEST FLAG FOR CONDITION
;
TSTFLG: ANI     38H     ;ISOLATE FLAG
        ORI     0C0H    ;ASSEMBLE A RETURN ON CONDITION INSTRUCTION
        STA     TST010  ;STORE FOR LATER EXECUTION
        LHLD    STSWRD  ;GET USER STATUS WORD
        PUSH    H       ;MOVE TO REAL FLAGS
        POP     PSW     ;
        MVI     A,1     ;A=1 MEANS CONDITION TRUE
TST010: RET             ;THIS IS REPLACED BY A CONDITIONAL RETURN
        MVI     A,0     ;CLEAR A IF CONDITION FALSE
        RET             ;
;
;CONDITIONAL JUMPS
;
JUMP:   LHLD    PC      ;GET USER PROGRAM COUNTER
        PUSH    H       ;SAVE IT
        PUSH    PSW     ;SAVE PSW
        MVI     B,3     ;THIS IS 3 BYTES
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        POP     PSW     ;GET PSW BACK
        PUSH    PSW     ;SAVE IT AGAIN
        LXI     H,#IHOP2        ;ADDRESS OF MNEUMONIC
        LXI     B,#IHOP2+1      ;ADDRESS TO PUT FLAG INTO
        CALL    GTFLAG          ;WHAT CONDITION?
        POP     PSW             ;GET INSTRUCTION BACK
        CALL    TSTFLG          ;TEST CONDITION
        POP     B               ;GET PC AGAIN
        ANA     A               ;CHECK RESULT OF FLAG TEST
        RZ              ;RETURN IF RESULT WAS NEGATIVE
        INX     H       ;UPDATE PC
        MOV     A,M     ;GET ADDRESS TO JUMP TO
        STA     PC      ;AND STORE TO USER'S PROGRAM COUNTER
        INX     H       ;
        MOV     A,M     ;
        STA     PC+1    ;
        RET             ;
;
;THIS ROUTINE PROCESSES ALL IMMEDIATE MODE COMMANDS
;
IMMEDT: MOV     C,A     ;SAVE INSTRUCTION
        MVI     B,2     ;ALL ARE 2 BYTES
        CALL    MVINST  ;MOVE INSTRUCTION TO EXECUTION AREA
        MOV     A,C     ;RESTORE INSTRUCTION
        ANI     38H     ;DETERMINE TYPE OF IMMEDIATE INSTRUCTION
        RRC             ;SCALE FOR INDEXING
        MOV     E,A     ;POSITION FOR INDEXING
        MVI     D,0     ;
        LXI     H,#IHOP6        ;GET BASE ADDRESS OF IMMEDIATE CODES
        DAD     D       ;ADD INDEX
        MOV     B,H     ;POSITION FOR SUBROUTINE CALL
        MOV     C,L     ;
        LXI     D,'  '  ;OTHER CHARACTERS ARE BLANK
        CALL    MVMNMC  ;DISPLAY MNEUMONIC
        CALL    OPEXEC  ;EXECUTE INSTRUCTION
        RET             ;
;
;MISCELANEOUS INSTRUCTIONS
;
MISC:   PUSH    PSW     ;SAVE INSTRUCTION
        ANI     38H     ;DETERMINE TYPE
        RRC             ;
        MOV     E,A     ;
        MVI     D,0     ;
        LXI     H,#IHOP1        ;
        DAD     D       ;
        MOV     C,L     ;
        MOV     B,H     ;
        LXI     D,'  '  ;
        CALL    MVMNMC  ;DISPLAY CODE
        POP     PSW     ;GET INSTRUCTION
        CPI     0E3H    ;
        JNC     XSDE    ;
        CPI     0D3H    ;
        JNC     IO      ;
        CPI     0CBH    ;
        JNC     NOOP    ;
        MVI     B,3     ;
        CALL    MVINST  ;
        LHLD    PC      ;
        DCX     H       ;
        MOV     A,M     ;
        STA     PC+1    ;
        DCX     H       ;
        MOV     A,M     ;
        STA     PC      ;
        RET             ;
NOOP:   LXI     B,#IHOP5+8
        LXI     D,'  '  ;
        CALL    MVMNMC  ;
        MVI     B,1     ;
        CALL    MVINST  ;
        RET             ;
IO:     MVI     B,2     ;
        CALL    MVINST  ;
        CALL    OPEXEC  ;
        RET             ;
XSDE:   MVI     B,1     ;
        CALL    MVINST  ;
        CALL    OPEXEC  ;
        RET             ;
;
;CONDITIONAL CALL ROUTINE
;
CLINST: PUSH    PSW     ;
        MVI     B,3     ;
        CALL    MVINST  ;
        POP     PSW     ;
        PUSH    PSW     ;
        LXI     B,#IHOP4        ;
        LXI     H,#IHOP4+1      ;
        CALL    GTFLAG  ;
        POP     PSW     ;
        CALL    TSTFLG  ;
        ANA     A       ;
        RZ              ;
CAL010: LHLD    PC      ;
        XCHG            ;
        LHLD    STKPTR  ;
        DCX     H       ;
        MOV     H,D     ;
        DCX     H       ;
        MOV     M,E     ;
        SHLD    STKPTR  ;
        LHLD    PC      ;
        DCX     B       ;
        MOV     A,M     ;
        STA     PC+1    ;
        DCX     B       ;
        MOV     A,M     ;
        STA     PC      ;
        RET             ;
;
;CONDITIONAL RETURN
;
RETURN: PUSH    PSW     ;
        MVI     B,1     ;
        CALL    MVINST  ;
        POP     PSW     ;
        PUSH    PSW     ;
        LXI     H,#IHOP         ;
        LXI     B,#IHOP+1       ;
        CALL    GTFLAG  ;
        POP     PSW     ;
        CALL    TSTFLG  ;
        ANA     A       ;
        RZ              ;
RET010: LHLD    STKPTR  ;
        MOV     A,M     ;
        STA     PC      ;
        INX     H       ;
        MOV     A,M     ;
        STA     PC+1    ;
        INX     H       ;
        SHLD    STKPTR  ;
        RET             ;
;
;MISCELANEOUS STACK INSTRUCTIONS
;
STSWRD: DB      '  B   D   H   PSW'
;
PPINST: PUSH    PSW     ;
        ANI     08H     ;
        JNZ     NOTPOP  ;
        LXI     B,#IHOP1        ;
POP010: LXI     D,4     ;
        LXI     H,INSTA ;
        CALL    SETLNE  ;
        POP     PSW     ;
        PUSH    PSW     ;
        ANI     '0'     ;
        RRC             ;
        RRC             ;
        MOV     C,A     ;
        MVI     B,0     ;
        LXI     H,STKWRD        ;
        DAD     B       ;
        MOV     B,H     ;
        MOV     C,L     ;
        LXI     H,INSTA+4       ;
        LXI     D,4     ;
        CALL    SETLNE  ;
        POP     PSW     ;
        MVI     B,1     ;
        CALL    MVINST  ;
        CALL    OPEXEC  ;
        RET             ;
NOTPOP: POP     PSW     ;
        CPI     0C9H    ;
        JNZ     NOTRET  ;
        MVI     B,1     ;
        CALL    MVINST  ;
        LXI     B,#IHOP1+6      ;
        LXI     D,'  '  ;
        CALL    MVMNMC  ;
        JMP     RET010  ;
NOTRET: CPI     0F9H    ;
        JNZ     NTSPHL  ;
        MVI     B,1     ;
        CALL    MVINST  ;
        LXI     B,#IHOP1+16     ;
        LXI     D,'  '  ;
        CALL    MVMNMC  ;
        LHLD    HL      ;
        SHLD    STKPTR  ;
        RET             ;
NTSPHL: CPI     0E9H    ;
        JNZ     NOOP    ;
        MVI     B,1     ;
        CALL    MVINST  ;
        LXI     B,#IHOP1+12     ;
        LXI     D,'  '  ;
        CALL    MVMNMC  ;
        LXI     H,HL    ;
        MOV     A,M     ;
        STA     PC      ;
```

```
        INX     H       ;
        MOV     A,M     ;
        STA     PC+1    ;
        RET             ;
;
;PUSH AND UNCONDITIONAL CALL
;
PSHNST: PUSH    PSW     ;
        ANI     08H     ;
        JNZ     CALLUN  ;
        LXI     B,HIHOP5        ;
        JMP     POPDIC  ;
CALLUN: POP     PSW     ;
        CPI     0CDH    ;
        JNZ     NOOP    ;
        LHLD    PC      ;
        PUSH    H       ;
        MVI     B,3     ;
        CALL    MVINST  ;
        LXI     B,HIHOP5+4      ;
        LXI     D,' '   ;
        LXI     H,INSTA ;
        CALL    MVMMMC  ;
        POP     B       ;
        JMP     CAL010  ;
;
;RESTARTS
;
RESTRT: PUSH    PSW     ;
        MVI     B,1     ;
        POP     PSW     ;
        PUSH    PSW     ;
        ANI     38H     ;
        RRC             ;
        RRC             ;
        RRC             ;
        CALL    HEXASC  ;
        MOV     D,C     ;
        MVI     B,' '   ;
        LXI     H,HIHOP7        ;
        CALL    MVMMMC  ;
        LHLD    PC      ;
        XCHG            ;
        LHLD    STKPTR  ;
        DCX     H       ;
        MOV     M,D     ;
        DCX     H       ;
        MOV     M,E     ;
        SHLD    STKPTR  ;
        POP     PSW     ;
        ANI     38H     ;
        MOV     C,A     ;
        MVI     H,0     ;
        SHLD    PC      ;
        RET             ;
;
;THE GO ROUTINE CONTROLS SIMULATED EXECUTION OF USER PROGRAMS
;
GO:     LHLD    CPOSIT  ;
        MVI     A,'-'   ;
        MOV     M,A     ;
        INX     H       ;
        SHLD    CPOSIT  ;
        CALL    GETADR  ;
        SHLD    PC      ;
        CALL    BMPLNE  ;
        LXI     B,STPADR        ;
        LXI     D,B     ;
        LXI     H,LINE15        ;
        CALL    SETLNE  ;
        LXI     H,LINS15+8      ;
        SHLD    CPOSIT  ;
        CALL    GETADR  ;
        CALL    BMPLNE  ;
        XCHG            ;
GO010:  LHLD    PC      ;
        MOV     A,D     ;
        SUB     H       ;
        JNZ     GO020   ;
        MOV     A,E     ;
        SUB     L       ;
        RZ              ;
GO020:  PUSH    D       ;
        PUSH    H       ;
        LHLD    INSTSP  ;
GO030:  DCX     H       ;
        MVI     D,10H   ;
GO040:  DCR     D       ;
        JNZ     GO040   ;
        MOV     A,H     ;
        ANA     A       ;
        JNZ     GO030   ;
        CALL    STEP    ;
        CALL    DSREGS  ;
        POP     H       ;
        POP     D       ;
        IN      KSTAT   ;
        ANI     KBDRDY  ;
        RZ      ;REPLACE WITH RNZ FOR BOARDS WITH INVERTED I/O STATUS
        JMP     GO010   ;
STPADR: DB      'STOP AT-'      ;
;
;THIS ROUTINE READS INTEL FORMAT TAPES
;
READTP: CALL    BMPLNE  ;
        CALL    TAPEIN  ;
        MOV     A,C     ;
        RZ              ;
        MVI     B,0     ;
        LXI     D,16    ;
        SUI     10H     ;
        MOV     C,A     ;
        LXI     H,ERRMES        ;
        DAD     B       ;
        MOV     B,H     ;
        MOV     C,L     ;
        LHLD    CPOSIT  ;
        CALL    BMPLNE  ;
        CALL    SETLNE  ;
        CALL    BMPLNE  ;
        RET             ;
;
;
;
TAPEIN: MVI     C,0     ;
        MVI     A,0FFH  ;
        OUT     SWTCHS  ;
BBLR:   CALL    INPUT   ;
        CPI     3AH     ;
        JNZ     BBLR    ;
        MVI     B,0     ;
        CALL    RDBYTE  ;
        MOV     D,A     ;
        CALL    RDBYTE  ;
        MOV     H,A     ;
        CALL    RDBYTE  ;
        MOV     L,A     ;
        CALL    RDBYTE  ;
        MOV     E,D     ;
        INR     E       ;
LD10:   DCR     E       ;
        JZ      LD20    ;
        MOV     A,L     ;
        CMA             ;
        OUT     SWTCHS  ;
        CALL    RDBYTE  ;
        MOV     M,A     ;
        CMP     M       ;
        INX     H       ;
        JZ      LD10    ;
        MVI     C,20H   ;
        RET             ;
LD20:   MOV     A,M     ;
        CMA             ;
        OUT     SWTCHS  ;
        CALL    RDBYTE  ;
        MOV     A,B     ;
        ORA     A       ;
        JZ      NBLK    ;
        MVI     C,10H   ;
        RET             ;
NBLK:   MOV     A,D     ;
        CMA             ;
        JNZ     BBLR    ;
        RET             ;
;
;
;
RDBYTE: PUSH    D       ;SAVE LENGTH
        CALL    INDIGT  ;GET 1 HEX DIGIT FROM TAPE
        ADD     A       ;MULTIPLY BY 16
        ADD     A       ;
        ADD     A       ;
        ADD     A       ;
        MOV     D,A     ;SAVE MSD
        CALL    INDIGT  ;GET NEXT HEX DIGIT FROM TAPE
        ORA     D       ;COMBINE HEX DIGITS TO FORM BYTE
        MOV     D,A     ;SAVE BYTE WHILE DOING CHKSUM
        ADD     B       ;ADD CHKSUM TO THE NEW BYTE
        MOV     B,A     ;REPLACE OLD CHKSUM WITH NEW
        MOV     A,D     ;GET NEW BYTE BACK
        POP     D       ;RESTORE LENGTH
        RET             ;
;                       ;
;
;
INDIGT: CALL    INPUT   ;READ A FRAME FROM TAPE
        CPI     '9'+1   ;INSPECT DATA
        JM      IND010  ;OK IF DATA < 10
        ADI     9       ;ELSE ADJUST FOR ASCII BIAS
IND010: ANI     0FH     ;REDUCE MOD 16
        RET             ;
;
;
INPUT:  IN      RSTAT   ;GET READER STATUS FROM SP+5
        ANI     RDRRDY  ;TURN OFF NON READER BITS
        JNZ     INPUT   ;LOOP UNTIL DATA AVAILABLE
        IN      READER  ;GET DATA
        ANI     7FH     ;CLEAR PARITY BIT
        RET             ;
;
;TAPE READER ROUTINE ERROR MESSAGES
;
ERRMES: DB      'CHECK SUM ERROR '
        DB      'MEMORY FAILURE  '
;
;THIS ROUTINE ZEROES A BLOCK OF MEMORY
;
ZERMEM: CALL    GTLMTS  ;
ZER010: MVI     A,0     ;
        MOV     M,A     ;
        INX     H       ;
        MOV     A,D     ;
        SUB     H       ;
        JNZ     ZERO10  ;
        MOV     A,E     ;
        SUB     L       ;
        JNZ     ZER010  ;
        RET             ;
;
;
;
;SPEED:  CALL    KEYBDT  ;
        CALL    ASCHEX  ;
        ADI     01H     ;
        MOV     H,A     ;
        MVI     L,0     ;
        MVI     A,100H  ;
        SUB     H       ;
        MOV     H,A     ;
        SHLD    INSTSP  ;
        CALL    BMPLNE  ;
        RET             ;
INSTSP: DW      0400H   ;
;
;
;
        END
```

Directory Program for CP/M Systems

Mark M. Zeiger

Like most computerists, I like things to be neat and orderly. I also like convenience, and having a large disk directory scroll off the screen before I can find what I'm looking for is not convenient. Therefore I was overjoyed when I discovered the CP/M Users Library had a program called **XDIR** that would output an alphabetized directory using the whole screen. I got a copy and literally ran home to try it out on my North Star CP/M system. Goodbye disk! I then tried it on a friend's eight inch double density system. While it didn't blow the disk, it also did not list the directory. Evidently the program was not CP/M compatible. However, once I had seen such a program, I had to have one for myself.

The program I have written is completely CP/M compatible. This means it does everything by using standard calls to the CP/M BDOS (which on most systems has its entry point at address 5 - if it doesn't, then it is not a "true" CP/M system). The only changes that have to be made from system to system are the commands that clear the screen and the tab control character (although the latter is pretty standard on most hardware).

The program has a number of "goodies"; the nicest being the Shell-Metzner sort. This sort is a fourteen line Basic program and it is not that much longer in assembly language. For a maximum of sixty-four items (the largest directory allowable in CP/M) almost any machine language sort would have been unnoticable timewise. As near as I can tell, the Shell-Metzner sort takes less than one-quarter of a second.

The maximum of sixty-four entries is perfect for a 80 x 24 screen. The heading and the line skipped after it along with the twenty lines of names will fit on the screen and still allow the CP/M prompt to be shown at the bottom without the screen scrolling. If there is a sixty-fourth entry, it will be shown at the bottom of the third column. If you would like to adapt the program to a VDM, it would be easy to do if there are not more than forty entries in the directory. More than that will defeat the purpose of the entire program unless you put a pause after sixteen lines are printed.

I did try to make the program structured, and at first it was very much so. But naturally, as a few more things were added, the structure started to disappear. Below are the major routines that are called sequentially at the beginning of the program and some of the more important subroutines:

CKDRIVE ========>	See if drive is requested, else use logged-in drive. Store drive name in heading.
SIGNON =========>	Print heading message.
GETNAME ========>	Checks to see if file name and/or type was requested; else finds all files.
CLEARBUFF ======>	Clears RAM where names are to be stored for sorting and output.
SEARCHRT =======>	Searches directory for names.
[- MOD4 ---------->	Finds address of directory FCB in DMA
[- TRANS --------->	Moves name and number of records to area in RAM where all names are to be stored contiguously.
EXTSCH =========>	Searches for file extents.
[	Notes the existence of extent, the number of extents, and the number of records in the last extent.
[	
[	
[	
[	
[- MATCH --------->	Searches buffer for matching 0th extent.
RECINDEC =======>	Divides by two to get the numbers of sectors (256 bytes). Then calculates total number of records in file in decimal. Puts number next to name with leading zeros surpressed.
[	
[	
[	
[	
[	
[	
[- ADDEXT --------->	Adds 64 decimal to number of sectors for each extent.
SORT ========>	Shell-Metzner Sort
[- COMPARE ------>	Compares the two names.
[- SWITCH ------->	Switches names in buffer if required.
PRINTOUT =======>	Prints names in three columns.
[	
[- WRITENAME ---->	Checks to see if name in directory buffer has been output to screen.

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

The only routine I will explain in detail is the "search" routine. When a search or "search next" is requested, CP/M loads the directory file control blocks into the DMA address (defaults in this program to 80H) in groups of four. These FCB's include files which have been erased as well as extents (which are not usually contiguous with the zeroth extent on the disk). The accumulator then returns a number which when divided by four will give a remainder that is the thirty-two byte part of the DMA address where the directory FCB is located. Thus, the remainders of 0, 1, 2, or 3 will correspond to 80H, 0A0H, 0C0H, 0E0H as the location of the FCB in the DMA address if the address is set at 80H. The MOD4 routine does this calculation. A 0FFH means the file does not exist. Extents have to be searched for as different files. Therefore, when first searching for the occurence of a file, the DE registers must point to a RAM address containing the name and extent of the file(s) being sought. The "search next" routine will then get other occurences of that file name (assuming, of course, that the filename is a wildcard). To search for extents, the DE registers must again point to the filename with the new extent and the initial search and the "search next(s)" must be requested.

I hope that you will enjoy the convience of this program as much as I have. One of the nice things about it is that it is slightly less than 1K of object code. This means that it will use the minimum amount of disk space (important for those of us who have mini-floppies). And also, is there anyone out there who knows how to calculate the amount of space left on a disk? The CP/M STAT program does it with a call to BDOS using 27 in the C-register, but I can not figure out the details of the routine. I would like to put it in this program. If anyone knows, I would appreciate hearing from you.

```
;CP/M DIRECTORY LIST PROGRAM

;COPYRIGHT 1979 BY MARK M. ZEIGER

;THIS PROGRAM WILL LIST AN ALPHABETIZED DIRECTORY OF A
;CP/M 1.4 DISK IN A FORMATTED OUTPUT ON A 80 X 24 SCREEN.
;NEXT TO EACH FILENAME IS THE NUMBER OF 256 BYTE PAGES
;IN THE FILE. THIS PROGRAM WILL WORK FOR ANY TYPE OF CP/M,
;WHETHER THE DISKS ARE IBM COMPATABLE FORMAT OR NOT, BECAUSE
;ALL DISK ACCESSES ARE DONE BY STANDARD CP/M FUNCTION CALLS.

;TO USE THE PROGRAM, JUST TYPE "XDIR". ALL FILES ON THE DEFAULT
;DRIVE WILL BE LISTED. IF YOU WISH TO EXAMINE ANOTHER DRIVE,
;SAY DRIVE B, TYPE "XDIR B:". IF YOU WISH TO LIST ONLY CERTAIN
;FILES, SUCH AS ALL COM FILES, TYPE "XDIR *.COM".

;REVISED 9/80 BY HARVEY FISHMAN TO WORK FOR CP/M 2 EXTENSION FORMATS
```

```
0100                    ORG 100H

0100 C32D01             JMP START

0005 =          BDOS    EQU 5
0011 =          SEARCH  EQU 17
0012 =          NXTSCH  EQU 18
0009 =          WRTBUF  EQU 9
0002 =          CONOUT  EQU 2
005C =          FCB     EQU 5CH
0015 =          NOLINES EQU 21

0103 3F         PRNTCNT DB 63
0104 00         DIRCNT  DB 0
0105            DESAVE  DS 2
0107            STKSV   DS 2
0109            WRTNUM  DS 1

  ;FCB FOR SEARCH ROUTINE. SEARCHES FOR
  ;ALL FILES UNLESS CHANGED BY GETNAME.

010A 003F3F3F   ANYNAME DB 0,'???????????',0,0,0,0,0
011B 00000000           DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

012D 210000     START   LXI H,0                ;SAVE STACK
0130 39                 DAD SP
0131 220701             SHLD STKSV
0134 319B05             LXI SP,NEWSTK
```

```
0137 CD5701            CALL CKDRIVE
013A CD7301            CALL SIGNON
013D CDCA01            CALL GETNAME
0140 CDB601            CALL CLEARBUFF
0143 CDDC01            CALL SEARCHRT           ;MAIN PROGRAM
0146 CD6202            CALL EXTSCH
0149 CDE602            CALL RECINDEC
014C CD4204            CALL SORT
014F CD6103            CALL PRINTOUT

0152 2A0701            LHLD STKSV              ;RELOAD CP/M'S STACK
0155 F9                SPHL
0156 C9                RET                     ;RETURN TO CP/M

0157 3A5C00    CKDRIVE LDA FCB                 ;GET DRIVE NUMBER IN FCB
015A 320A01            STA ANYNAME
015D FE00              CPI 0                   ;IF DRIVE IS ZERO, THEN..
015F CA6801            JZ LOGDSK               ;..CALCULATE LOGGED-IN DRIVE.
0162 C640              ADI 40H                 ;CHANGE TO ASCII
0164 329701            STA DRMSG
0167 C9                RET
0168 0E19      LOGDSK  MVI C,25                ;CP/M GET CURRENT DRIVE CALL
016A CD0500            CALL BDOS
016D C641              ADI 41H                 ;CHANGE TO ASCII
016F 329701            STA DRMSG
0172 C9                RET

0173 117A01    SIGNON  LXI D,ONMSG             ;SCREEN CLEAR AND..
0176 CDFB03            CALL WRITOUT            ;..PRINT HEADING.
0179 C9                RET

017A 1A0C2044  ONMSG   DB 1AH, 0CH,' Directory      ',9,'    ','Drive '

0197 =         DRMSG   EQU $-1

0198 0909436F          DB 9,9, 'Copyright 1979  M. Zeiger',0DH,0AH,0

CLEARBUFF

  ;DIRECTORY BUFFER FILLED WITH SPACES. BUFFER IS 5 PAGES
  ;AT TOP OF PROGRAM.

01B6 3E20              MVI A,' '
01B8 219C05            LXI H,DIRBUFF
01BB 1605              MVI D,5
01BD 1E00      LP1     MVI E,0
01BF 77        LP2     MOV M,A
01C0 23                INX H
01C1 1C                INR E
01C2 C2BF01            JNZ LP2
01C5 15                DCR D
01C6 C2BD01            JNZ LP1
01C9 C9                RET

GETNAME

  ;PUTS FILE NAME AMD/OR TYPE INTO SEARCH FCB. IF DEFAULT FCB
  ;OF COMMAND LINE IS BLANK, THEN LEAVE SEARCH FCB WITH "?"S.

01CA 3A5D00            LDA FCB+1
01CD FE20              CPI ' '
01CF C8                RZ
01D0 215D00            LXI H,FCB+1
01D3 110B01            LXI D,ANYNAME+1
01D6 060B              MVI B,11
01D8 CDF904            CALL HTD
01DB C9                RET

SEARCHRT

  ;SEARCHES FOR NAMES AND TRANSFERS TO BUFFER ABOVE PROGRAM.

01DC 219C05            LXI H,DIRBUFF           ;SAVE ADDRESS..
01DF 220501            SHLD DESAVE             ;..OF DIRBUFF
01E2 0E11              MVI C,SEARCH            ;SEARCH FOR FIRST..
01E4 110A01            LXI D,ANYNAME           ;..OCCURANCE.
01E7 CD0500            CALL BDOS
01EA FEFF      CHECK1  CPI 0FFH                ;IF FIRST SEARCH FAILS..
01EC CA1302            JZ NODIR                ;..NO ENTRY EXISTS.

  ;FIND NEXT VALID FILENAME IN DMA. MULTIPLY IT BY 32, THE LENGTH
  ;OF A DISK DIRECTORY. CALCULATE ITS ADDRESS IN DMA IN HL REG.

01EF CD5502    LOOP2   CALL MOD4
01F2 E5                PUSH H                  ;ZERO THE 3 BYTES AFTER NAME
01F3 D5                PUSH D
01F4 110C00            LXI D,12
01F7 19                DAD D
01F8 3600              MVI M,0
01FA 23                INX H
01FB 3600              MVI M,0
01FD 23                INX H
01FE 3600              MVI M,0
0200 D1                POP D
0201 E1                POP H
0202 CD3402            CALL TRANS              ;TRANSFER TO DIRBUFF
0205 0E12              MVI C,NXTSCH            ;SEARCH FOR NEXT ENTRY
0207 110A01            LXI D,ANYNAME
020A CD0500            CALL BDOS
020D FEFF              CPI 0FFH                ;NO MORE NAMES IF 0FFH
020F C2EF01            JNZ LOOP2
0212 C9                RET

0213 112002    NODIR   LXI D,NODIRMSG          ;COULDN'T FIND THE ENTRY
0216 0E09              MVI C,WRTBUF
0218 CD0500            CALL BDOS
021B 2A0701            LHLD STKSV
021E F9                SPHL
021F C9                RET

0220 0D0A0A4E  NODIRMSG DB 0DH,0AH,0AH,'No entry found',0DH,0AH,'$'

TRANS

  ;STORES A DISK FCB ALONG WITH NUMBER OF RECORDS IN THE
  ;NEXT 16 BYTES OF DIRBUFF.

0234 F5                PUSH PSW
0235 D5                PUSH D
0236 0610              MVI B,16
0238 E5                PUSH H                  ;GET THE NEXT ADDRESS..
0239 2A0501            LHLD DESAVE             ;..OF FILE IN DIRBUFF.
023C EB                XCHG                    ;PUT IT INTO DE REG
023D E1                POP H
023E 7E        LOOP1   MOV A,M                 ;DO THE TRANSFER
023F 12                STAX D
0240 23                INX H
0241 13                INX D
0242 05                DCR B
0243 C23E02            JNZ LOOP1
0246 EB                XCHG                    ;SAVE THE LAST ADDRESS..
0247 220501            SHLD DESAVE             ;..OF DIRBUFF USED.
024A EB                XCHG
024B D1                POP D
```

```
024C 3A0401            LDA DIRCNT          ;COUNT THE NUMBER OF..
024F 3C                INR A               ;..DIRECTORY ENTRIES.
0250 320401            STA DIRCNT
0253 F1                POP PSW
0254 C9                RET

MOD4

   ;CALCULATE THE ADDRESS OF THE DIRECTORY FCB IN DMA

0255 E603              ANI 03H             ;GET DIR ENTRY MODULO 4
0257 070707            RLC ! RLC ! RLC     ;MULT BY 32
025A 0707              RLC ! RLC
025C 218000            LXI H,80H           ;GET ADDR IN DMA (80H)
025F 85                ADD L
0260 6F                MOV L,A
0261 C9                RET

EXTSCH

   ;SEARCH FOR ALL FILES WITH EXTENT 1

0262 211601            LXI H,ANYNAME + 12  ;INCREASE EXTENT NUMBER
0265 34                INR M
0266 0E11              MVI C,SEARCH
0268 110A01            LXI D,ANYNAME
026B CD0500            CALL BDOS
026E FEFF              CPI 0FFH            ;IF THERE IS NOT A FIRST EXTENT..
0270 C8                RZ                  ;..THEN DONE WITH SEARCHES.

0271 47                MOV B,A             ;CODE TO NEXT BLANK LINE..
0272 3A1601            LDA ANYNAME+12      ;..ADDED FOR CP/M 2 BY..
0275 E601              ANI 1               ;..HARVEY FISHMAN.
0277 78                MOV A,B
0278 CA8802            JZ LOOP5
027B CD5502            CALL MOD4
027E C60C              ADI 12
0280 6F                MOV L,A
0281 7E                MOV A,M
0282 E601              ANI 1
0284 CAA802            JZ NXT1
0287 78                MOV A,B             ;END OF CODE BY H.F.

0288 CD5502    LOOP5   CALL MOD4           ;GET ADDR IN HL
028B EB                XCHG                ;PUT IT IN DE.
028C 21AC05            LXI H,DIRBUFF
028F CDCF02    LOOP4   CALL MATCH          ;COMPARES EXTENTS WITH 0TH EXT.
0292 DA9C02            JC FOUNDMATCH       ;ITS ADDRESS IS IN HL REG
0295 011000            LXI B,16
0298 09                DAD B               ;TEST THE NEXT ADDRESS IN DIRBUFF
0299 C38F02            JMP LOOP4

FOUNDMATCH

029C EB                XCHG                ;PUT DMA ADDRESS IN HL REG
029D 010F00            LXI B,15
02A0 09                DAD B
02A1 EB                XCHG                ;DE POINTS TO # RECS IN EXT ENTRY
02A2 09                DAD B               ;HL POINTS TO # RECORDS IN DIRBUFF
02A3 1A                LDAX D              ;GET # RECORDS IN EXTENT
02A4 77                MOV M,A             ;PUT IT IN # RECORDS OF DIRBUFF
02A5 2B                DCX H               ;POINT HL TO A BYTE TO BE..
02A6 2B                DCX H               ;..DESIGNATED AS NUMBER OF EXTS.
02A7 34                INR M               ;COUNT NUMBER OF EXTENTS
02A8 0E12      NXT1    MVI C,NXTSCH        ;SEARCH FOR NEXT FILE
02AA 110A01            LXI D,ANYNAME

02AD CD0500            CALL BDOS
02B0 FEFF              CPI 0FFH

02B2 CA6202            JZ EXTSCH           ;CODE TO NEXT BLANK LINE ADDED..
02B5 47                MOV B,A             ;..FOR CP/M 2 BY HARVEY FISHMAN.
02B6 3A1601            LDA ANYNAME+12
02B9 E601              ANI 1
02BB 78                MOV A,B
02BC CA8802            JZ LOOP5
02BF CD5502            CALL MOD4
02C2 C60C              ADI 12
02C4 6F                MOV L,A
02C5 7E                MOV A,M
02C6 E601              ANI 1
02C8 CAA802            JZ NXT1
02CB 78                MOV A,B
02CC C38802            JMP LOOP5           ;END OF CODE BY H.F.

MATCH

   ;CALLED WITH DE POINTING TO FILENAME IN FCB AND HL POINTING
   ;TO FILE NAME IN DIRBUFF. RETURNS WITH CARRY SET IF THE
   ;NAMES ARE THE SAME.

02CF E5                PUSH H
02D0 D5                PUSH D
02D1 0E0B              MVI C,11
02D3 1A        LOOP3   LDAX D
02D4 BE                CMP M
02D5 C2E202            JNZ CLRCRY          ;THEY'RE NOT EQUAL, CLEAR CARRY
02D8 23                INX H               ;CHECK NEXT CHARACTERS
02D9 13                INX D
02DA 0D                DCR C
02DB C2D302            JNZ LOOP3           ;CHECK 11 CHARACTERS.
02DE D1                POP D               ;THEY'RE EQUAL
02DF E1                POP H
02E0 37                STC
02E1 C9                RET
02E2 B7        CLRCRY  ORA A               ;CLEAR CARRY
02E3 D1                POP D
02E4 E1                POP H
02E5 C9                RET

RECINDEC

   ;GETS DECIMAL NUMBER OF PAGES (256 BYTES) IN FILE EXTENTS.

02E6 3A0401            LDA DIRCNT
02E9 47                MOV B,A             ;NUMBER OF DIRECTORY ENTRIES IN A
02EA 21AB05            LXI H,DIRBUFF + 15  ;POINT TO NUMBER OF RECORDS..
02ED 7E        LOOP8   MOV A,M             ;..IN FIRST DIRBUFF ENTRY.
02EE 3C                INR A               ;INCREASE BY 1, THEN DIVIDE BY 2..
02EF B7                ORA A               ;..TO CHANGE RECORDS TO PAGES.
02F0 1F                RAR
02F1 CD2B03            CALL DECIMAL        ;CHANGE BINARY TO DECIMAL
02F4 2B                DCX H               ;POINT TO NUMBER OF..
02F5 2B                DCX H               ;..EXTENTS PER FILE.
02F6 7E                MOV A,M
02F7 CD3B03            CALL ADDEXT         ;RETURNS WITH UNITS IN E..
02FA 2B                DCX H               ;..TENS IN D, HUNDREDS IN C.
02FB 79                MOV A,C
02FC FE00              CPI 0               ;BLANK IF 100'S IS ZERO
02FE C21503            JNZ SKIP4           ;MAKE IT A DIGIT
0301 F620              ORI 20H             ;MAKE IT A BLANK
0303 77                MOV M,A             ;PUT IT BACK IN DIRBUFF
0304 23                INX H               ;IF HUNDRED'S ARE BLANK..
```

```
0305 7A              MOV A,D             ;..TEST TO SEE IF TEN'S..
0306 FE00            CPI 0               ;..SHOULD BE BLANK.
0308 C21003          JNZ SKIP9
030B F620            ORI 20H
030D C31C03          JMP SKIP10
0310 F630     SKIP9  ORI 30H
0312 C31C03          JMP SKIP10
0315 F630     SKIP4  ORI 30H             ;MAKE IT ASCII
0317 77       SKIP5  MOV M,A
0318 23              INX H
0319 7A              MOV A,D             ;TEN'S DIGIT
031A F630            ORI 30H             ;MAKE IT ASCII
031C 77       SKIP10 MOV M,A
031D 23              INX H
031E 7B              MOV A,E
031F F630            ORI 30H
0321 77              MOV M,A
0322 111100          LXI D,17            ;READY FOR NEXT ENTRY..
0325 19              DAD D               ;..IN DIRBUFF.
0326 05              DCR B
0327 C2ED02          JNZ LOOP8
032A C9              RET

DECIMAL

   ;CHANGES A-REG TO DECIMAL. RETURNS WITH TEN'S IN D,
   ;UNIT'S IN E.

032B 110000          LXI D,0
032E F5              PUSH PSW
032F D60A     LOOP9  SUI 10
0331 14              INR D               ;COUNT NUMBER OF TENS
0332 D22F03          JNC LOOP9
0335 15              DCR D
0336 C60A            ADI 10
0338 5F              MOV E,A             ;UNIT'S DIGIT IN E
0339 F1              POP PSW
033A C9              RET

ADDEXT

   ;FOR EVERY EXTENT, ADD 64 DECIMAL TO NUMBER OF PAGES.
   ;HUNDRED'S IN C, TEN'S IN D, AND UNIT'S IN E.

033B 0E00            MVI C,0
033D FE00            CPI 0               ;IF NO EXTENTS, EXIT
033F C8              RZ
0340 F5              PUSH PSW            ;SAVE NUMBER OF EXTENTS
0341 7A              MOV A,D             ;PUT TEN'S IN A(REG)
0342 0707            RLC ! RLC           ;GET IT IN UPPER NIBBLE
0344 0707            RLC ! RLC
0346 E6F0            ANI 0F0H            ;BLANK OUT LOWER NIBBLE
0348 B3              ORA E               ;GET UNIT'S IN LOWER NIBBLE
0349 C664            ADI 64H             ;EACH EXTENT IS 64 PAGES
034B 27              DAA
034C D25003          JNC SKIP2
034F 0C              INR C               ;COUNT HUNDREDS PLACE
0350 F5       SKIP2  PUSH PSW
0351 E60F            ANI 0FH             ;BLANK OUT TEN'S
0353 5F              MOV E,A             ;MOVE UNIT'S TO E
0354 F1              POP PSW
0355 0F0F            RRC ! RRC
0357 0F0F            RRC ! RRC           ;PUT TEN'S IN LOWER NIBBLE
0359 E60F            ANI 0FH             ;BLANK OUT UPPER NIBBLE
035B 57              MOV D,A             ;PUT TEN'S IN D
035C F1              POP PSW             ;GET BACK NUMBER OF..
035D 3D              DCR A               ;..EXTENTS AND DECREASE.
035E C33D03          JMP ADDEXT + 2

PRINTOUT

   ;WRITES DIRBUFF TO 80 X 24 SCREEN. WRITES IN 3 COLUMNS. IF
   ;64 ENTRIES, WRITES 64TH AT BOTTOM OF THIRD COLUMN. WRITES
   ;THE 1ST, 22ND, 43RD RECORD, THEN THE 2ND, 23RD, 44TH RECORD,
   ;ETC. IF THE RECORD HAS BEEN BLANKED IN DIRBUFF BECAUSE IT
   ;HAS ALREADY BEEN WRITTEN OR THERE ARE NO MORE RECORDS, IT
   ;DOES NOT GET WRITTEN.

0361 3A0401          LDA DIRCNT
0364 320901          STA WRTNUM
0367 CD2404          CALL CRLF
036A 3E00            MVI A,0
036C 321605   LOOP6  STA RECNO
036F CDB103          CALL WRITENAME
0372 C615            ADI NOLINES
0374 CDB103          CALL WRITENAME
0377 C615            ADI NOLINES
0379 CDB103          CALL WRITENAME
037C 3A0301          LDA PRNTCNT
037F D603            SUI 3
0381 320301          STA PRNTCNT
0384 FE00            CPI 0               ;CHECK FOR 64TH ENTRY
0386 CA9803          JZ PRINTEND
0389 3A0901          LDA WRTNUM          ;IF ALL ENTRIES ARE..
038C B7              ORA A               ;..PRINTED, RETURN.
038D C8              RZ
038E CD2404          CALL CRLF
0391 3A1605          LDA RECNO
0394 3C              INR A
0395 C36C03          JMP LOOP6

PRINTEND                 ;MUST GET THE 64TH ENTRY..

0398 3A0401          LDA DIRCNT          ;..IF THERE IS ONE.
039B FE40            CPI 64
039D C0              RNZ
039E 113A04          LXI D,TAB7          ;TAB TO LAST COLUMN FOR LAST ENTRY
03A1 0E09            MVI C,WRTBUF
03A3 CD0500          CALL BDOS
03A6 3E00            MVI A,0             ;STIFFLE ANYMORE TABS
03A8 322204          STA NULLIT
03AB 3E3F            MVI A,63
03AD CDB103          CALL WRITENAME
03B0 C9              RET

WRITENAME

   ;CHANGES "RECNO" TO (DIRBUFF+1) + 16*RECON. IT THEN STORES
   ;THAT ADDRESS IN HL AND PRINTS THE NAME AT THAT ADDRESS
   ;UNLESS IT IS BLANK.

03B1 C5              PUSH B
03B2 019D05          LXI B,DIRBUFF+1
03B5 6F              MOV L,A
03B6 2600            MVI H,0
03B8 2929            DAD H ! DAD H
03BA 2929            DAD H ! DAD H
03BC 09              DAD B

03BD F5              PUSH PSW
03BE 7E              MOV A,M             ;TEST TO SEE IF NAME WAS..
03BF FE20            CPI ' '             ;..PREVIOUSLY OVERWRITTEN.
```

```
03C1 CAED03          JZ NOWRITE
03C4 3A0901          LDA WRTNUM
03C7 3D              DCR A
03C8 320901          STA WRTNUM
03CB 0E08            MVI C,8         ;TRANSFER NAME TO OUTPUT BUFFER
03CD 110C04          LXI D,OUTBUFF
03D0 CDF003          CALL TRANS2
03D3 3E20            MVI A,' '       ;AT LEAST ONE SPACE BETWEEN..
03D5 12              STAX D          ;..NAME AND TYPE.
03D6 13              INX D
03D7 0E03            MVI C,3         ;TRANSFER TYPE
03D9 CDF003          CALL TRANS2
03DC 3E20            MVI A,' '
03DE 12              STAX D
03DF 13              INX D
03E0 12              STAX D
03E1 13              INX D
03E2 0E03            MVI C,3
03E4 CDF003          CALL TRANS2
03E7 110C04          LXI D,OUTBUFF
03EA CDFB03          CALL WRITOUT
03ED F1      NOWRITE POP PSW
03EE C1              POP B
03EF C9              RET

03F0 7E      TRANS2  MOV A,M         ;TRANSFER ROUTINE TO OUTPUT BUFFER
03F1 12              STAX D
03F2 3620            MVI M,' '       ;OVERWRITE NAME WITH BLANKS
03F4 23              INX H
03F5 13              INX D
03F6 0D              DCR C
03F7 C2F003          JNZ TRANS2
03FA C9              RET

03FB 1A      WRITOUT LDAX D          ;WRITES OUT EACH CHAR..
03FC B7              ORA A           ;..INDIVIDUALLY SINCE USING..
03FD C8              RZ              ;..CP/M'S PRINT STRING..
03FE C5              PUSH B          ;..ROUTINE WILL NOT PRINT $'S..
03FF D5              PUSH D          ;..WHICH ARE SOMETIMES FILETYPES.
0400 0E02            MVI C,CONOUT
0402 5F              MOV E,A
0403 CD0500          CALL BDOS
0406 D1              POP D
0407 C1              POP B
0408 13              INX D
0409 C3FB03          JMP WRITOUT

040C         OUTBUFF DS 17
041D 20207C20        DB ' | '        ;SEPARATER OF FILE NAMES..
0422 0900    NULLIT  DB 9,0          ;..FOR EACH COLUMN.

0424 F5      CRLF    PUSH PSW
0425 D5              PUSH D
0426 C5              PUSH B
0427 E5              PUSH H
0428 113704          LXI D,CARLFD
042B 0E09            MVI C,WRTBUF
042D CD0500          CALL BDOS
0430 E1              POP H
0431 C1              POP B
0432 D1              POP D
0433 F1              POP PSW
0434 0603            MVI B,3
0436 C9              RET

0437 0D0A24  CARLFD  DB 0DH,0AH,'$'

043A 090909  TAB7    DB 9,9,9,9,9,9,9,'$'

             SORT
               ;THE FOLLOWING IS AN ADAPTION OF THE SHELL-METZNER SORT
               ;TAKEN FROM A BASIC PROGRAM.
0442 3A0401          LDA DIRCNT      ;N = M = NUMBER OF ITEMS
0445 321605          STA RECNO
0448 321705          STA HALFREC
044B 3A1705  HALVE   LDA HALFREC     ;M = INT(M/2)
044E B7              ORA A
044F 1F              RAR
0450 321705          STA HALFREC
0453 C8              RZ              ;M = 0 ?  YES - EXIT SORT
0454 3A1705          LDA HALFREC     ;K = N - M
0457 47              MOV B,A
0458 3A1605          LDA RECNO
045B 90              SUB B
045C 321805          STA SPREAD
045F 3E00            MVI A,0         ;J = 0
0461 321B05          STA J
0464 3A1B05  X2      LDA J           ;I = J
0467 321905          STA FIRSTREC
046A 3A1905  X1      LDA FIRSTREC    ;L = I + M
046D 47              MOV B,A
046E 3A1705          LDA HALFREC
0471 80              ADD B
0472 321A05          STA SECONDREC
0475 CDA404          CALL CHTOAD     ;CHANGE RECORD TO ADDRESS IN DIRBUFF
0478 CDBF04          CALL COMPARE    ;D(I) > D(J) ?
047B D29204          JNC SKIPTRANS   ;IF NO CARRY SET, THEN DO NOT SWITCH
047E CDD904          CALL SWITCHHD   ;ELSE MAKE THE SWITCH
0481 3A1705          LDA HALFREC     ;I = I - M
0484 47              MOV B,A
0485 3A1905          LDA FIRSTREC
0488 90              SUB B
0489 321905          STA FIRSTREC
048C FA9204          JM SKIPTRANS    ;IF I < 0 THEN LOOP BACK
048F C36A04          JMP X1

             SKIPTRANS
0492 3A1B05          LDA J           ;J = J + 1
0495 3C              INR A
0496 321B05          STA J
0499 47              MOV B,A         ;IS J > K
049A 3A1805          LDA SPREAD
049D B8              CMP B
049E DA4B04          JC HALVE        ;IF J > K GOTO 'HALVE'
04A1 C36404          JMP X2          ;IF J <= K GOTO 'X2'

             CHTOAD
               ;CHANGES "RECNO" TO CORRECT ADDRESS IN DIRBUFF. HL POINTS
               ;TO FIRST RECORD AND DE POINTS TO SECOND.
04A4 3A1905          LDA FIRSTREC
04A7 CDB304          CALL ADJUST
04AA EB              XCHG
04AB 3A1A05          LDA SECONDREC
04AE CDB304          CALL ADJUST
04B1 EB              XCHG
04B2 C9              RET
```

```
ADJUST

   ;MULTIPLIES "RECNO" BY 16 AND PUTS IN HL TO POINT
   ;TO NAME IN DIRBUFF.

 04B3 019C05            LXI B,DIRBUFF
 04B6 6F                MOV L,A
 04B7 2600              MVI H,0
 04B9 2929              DAD H ! DAD H
 04BB 2929              DAD H ! DAD H
 04BD 09                DAD B
 04BE C9                RET

COMPARE

   ;COMPARES THE NAMES IN THE FIRST AND SECOND ADDRESS. IF THE
   ;FIRST IS LARGER THAN THE SECOND, IT INDICATES A SWITCH
   ;SHOULD BE MADE BY SETTING THE CARRY.

 04BF E5                PUSH H
 04C0 D5                PUSH D
 04C1 0E0B              MVI C,11
 04C3 46                MOV B,M
 04C4 1A                LDAX D
 04C5 B8                CMP B
 04C6 DACD04            JC RETWC
 04C9 CAD004            JZ INCREASE
 04CC B7                ORA A               ;CLEAR CARRY
 04CD D1        RETWC   POP D
 04CE E1                POP H
 04CF C9                RET

INCREASE                        ;CHARACTERS WERE EQUAL. DO..

 04D0 23                INX H                ;..ANOTHER COMPARE.
 04D1 13                INX D
 04D2 0D                DCR C
 04D3 CACB04            JZ RETWC-2
 04D6 C3C304            JMP COMPARE+4

SWITCHHD

   ;THIS ROUTINE SWITCHES THE FIRST RECORD WITH THE SECOND.

 04D9 D5                PUSH D
 04DA E5                PUSH H
 04DB 110605            LXI D,TEMP          ;PUT THE SECOND RECORD IN..
 04DE 0610              MVI B,16            ;.. TEMPORARY STORAGE.
 04E0 CDF904            CALL HTD
 04E3 E1                POP H               ;PUT THE FIRST RECORD..
 04E4 D1                POP D               ;..IN THE SECOND.
 04E5 EB                XCHG
 04E6 0610              MVI B,16
 04E8 CDF904            CALL HTD
 04EB EB                XCHG                ;PUT THE TEMPORARY (SECOND)..
 04EC E5                PUSH H              ;..IN THE FIRST.
 04ED D5                PUSH D
 04EE 210605            LXI H,TEMP
 04F1 0610              MVI B,16
 04F3 CDF904            CALL HTD
 04F6 D1                POP D
 04F7 E1                POP H
 04F8 C9                RET

HTD

   ;THIS ROUTINE DOES THE TRANSFER. MOVES RECORD ADDRESSED BY
   ;HL TO RECORD ADDRESSED BYE DE.

 04F9 D5                PUSH D
 04FA E5                PUSH H
 04FB 7E                MOV A,M
 04FC 12                STAX D
 04FD 23                INX H
 04FE 13                INX D
 04FF 05                DCR B
 0500 C2FB04            JNZ HTD+2
 0503 E1                POP H
 0504 D1                POP D
 0505 C9                RET

 0506           TEMP    DS 16
 0516           RECNO   DS 1
 0517           HALFREC DS 1
 0518           SPREAD  DS 1
 0519           FIRSTREC DS 1
 051A           SECONDREC DS 1
 051B           J       DS 1

 051C           STACK   DS 80H
 059B =         NEWSTK  EQU $-1

 059C           DIRBUFF DS 16*64

 099C                   END
```

Modification to CBasic 2

Ben and Andy Galewsky

CBasic by Software Systems is a good language for many applications, especially in the business environment. The CBasic language comes as a package of two programs. The Basic source is entered into a file using a text editor, then compiled into intermediate code by the program CBAS2. The intermediate code is executed by invoking CRUN2.

Unfortunately, the language has one major shortcoming. There is no provision for outputing a single character to the console at the current cursor position; a buffer must be filled and then printed. This creates problems for users with memory mapped video displays. Formated screens and other special programs also become difficult (i.e. Osborne and Associates' Payroll with Cost Accounting).

It is possible to write a machine language subroutine to output a single character and have CBasic load the program every time it is run. This has its own attendant problems. The solution presented in this article is a modification to CRUN2. A machine language subroutine is inserted into an unused portion of CRUN2. The character to be placed on the screen is POKEd into a memory location specified by the subroutine. The subroutine is then CALLed from Basic. Then the subroutine makes a call to CP/M to display the character at the cursor position. This eliminates the need to load the routine from disk every time the program is run because it travels along with CRUN2.

The second modification involves the CRUN2 sign-on message, allowing a more elegant and custom finish, as well as making computer operation easier for an inexperienced user.

Making The Modification

Before attempting to modify any program, especially expensive or irreplaceable software, a copy should be made and kept in a safe place free from magnetic radiation and high temperature.

Galewsky, 7705 Calder Ave, Beaumont, TX 77706

With the backup made, it is now possible to begin the modifications. For this you will need to use DDT (Dynamic Debugging Tool) supplied with your CP/M system, or a similar program. First invoke DDT by typing DDT CRUN2.COM. DDT will return with the following prompt:

```
DDT VERS 1.4
NEXT PC
4300 0100
```

The 4300 under the NEXT shows the next available address after CRUN2. The 100 under PC tells the location of the program counter.

Starting around 110 hex is an embedded copyright notice. This area can be displayed by typing D100 (figure 1). It is here that the new machine language subroutine will be placed. To load the program into memory, the in-memory assembly function of DDT will be used. Type A120. to start the assembly at 120 hex. Type in the following program:

```
120 MVU C,02
122 LDA 130
125 MOV E,A
129 CALL 0005
129 RET
```

Key <RETURN> to end the in-memory assembly function. When called, this program loads the CP/M code for print character (2) in to the C register of the microprocessor. Then it fetches the character out of memory location 130 hex and moves it to the E register to be passed to CP/M. Finally CP/M is called at 0005 hex to place the character on the screen. This solves the single character output problem.

The next modification is to the sign-on message. This message is found at 2147 hex (on CRUN vs 2.05). Display this message by typing D2100 (figure 2). The new message may be up to 18 characters long including a

terminal dollar sign (the extra dollar signs may be written over). In our case we decided to have CRUN2 clear the screen and print

Please wait...

Which is a little less confusing and more reassuring to the inexperienced operator then the usual

CRUN VER 2.05

message. The revised message is shown in Figure 3. To put the proper characters into memory, use the DDT S command. This displays the memory contents and allows you to change it. Type S2147 and key in the proper ASCII codes (see figure 4). The 04 code at the begining is the screen clear code for the Vector Mindless Terminal. Use whatever screen clear character you particular terminal uses. The remaining codes are for the Please wait... message. End the message with a "$" (ASCII code 24). The dollar sign is the terminator of a message string used by CP/M. Type <Q> to Quit the change mode. Display the message again with the command D2147 to check for proper coding (see figure 4). This ends the modification.

The modified CRUN2 must be saved on the disk. To do this type control C. This does a warm start and returns to the A> prompt. Type SAVE 72 RUN.COM. This saves 72 256 byte pages into the file RUN.COM We use the name RUN.COM to make programs easier to run. The operator only has to type RUN > filename.

Testing The Modifications

As with any program, all changes must be thoroughly tested. Testing the sign-on message is easily done; simply run any CBasic program and your sign-on message should be displayed in lieu of CRUN VS 2.OX

To test the single character printing, a short program will have to be written. The program in listing 1 is an example. This program uses the CONCHAR% function of CBasic. It will input a line of characters and then allow the editing of this line. The functions supported are:

<space bar>	advances to the next character
D	deletes the current character
C	changes the current character
<return>	inputs a new line to edit

This program is quite useful as an editor of input data in a program. Function PRT uses the single character print routine to display the argument DISP$.

In Conclusion

These modifications overcome some of CBasic's limitations. Combining these changes with the turnkey CP/M system described in December, 1979 *Creative Computing* will aid in the operation of your application programs.

—PROGRAM ON NEXT PAGE—

```
-D2100
0100 C3 A0 26 C3 00 00 00 00 0E 0D C3 05 00 C3 00 00 ..&.............
0110 50 00 43 4F 50 59 52 49 47 48 54 20 28 43 29 20 P.COPYRIGHT (C)
0120 31 39 37 37 20 20 31 39 37 38 20 20 31 39 37 39 1977,  1978,  1979
0130 20 43 4F 4D 50 49 4C 45 52 20 53 59 53 54 45 4D  COMPILER SYSTEM
0140 53 20 49 4E 43 05 00 00 00 00 00 00 00 00 00 00 S INC.....
0150 00 00 00 00 00 00 00 00 00 00 E1 E3 78 B1 C8 1A ..........x...
0160 77 0B 13 23 23 5C 01 0A 12 03 13 0A 12 03 13 0A w..#.\.....
0170 12 03 13 0A 12 03 13 0A 12 03 13 0A 12 03 13 0A ................
0180 12 03 13 0A 12 C9 21 8B 01 71 DB 00 C9 21 93 01 ......!..q...!..
0190 71 7B D3 00 C9 60 69 E9 21 B2 01 4E 23 46 2B CD q{...`i.!..N#F+.
01A0 B6 01 21 B4 01 CD D3 01 21 B2 01 71 23 7B E6 7F ..!.....!..q#x..
01B0 77 C9 00 00 19 36 16 09 CD C7 01 CD D3 01 16 02 w....6..........
-
```

Figure 1

```
D2100
2100 0F 0F 0F E6 60 C6 0F 2A 05 1A 4F 06 00 09 7E C6 ....`..*..O...~.
2110 07 0F 0F 0F E6 1F 4F E1 09 E5 0E 12 C3 F8 20 CD ......O....... .
2120 5E 1A E1 AF 32 1A 1A C9 4E 4F 20 49 4E 54 45 52 ^...2...NO INTER
2130 4D 45 44 49 41 54 45 20 4C 41 4E 47 55 41 47 45 MEDIATE LANGUAGE
2140 20 46 49 4C 45 20 24 43 52 55 4E 20 56 45 52 20  FILE $CRUN VER
2150 32 2E 30 35 24 24 24 24 24 24 24 24 21 00 01 22 24 2.05$$$$$$$$!..."$
2160 41 21 50 00 22 26 41 3A 45 41 1F DA 86 21 CD E6 A!P."&A:EA...!..
2170 17 01 47 21 CD 1A 27 0E 0D CD C4 26 0E 0A CD C4 ..G!..'....&....
2180 26 21 EF 01 36 00 CD CC 24 21 0E 43 22 FC 40 CD &!..6...$!.C".@.
2190 5D 25 4F 3E 02 B9 D2 A2 21 01 56 49 CD 1D 28 CD ]%O>....!.VI..(.
21A0 0D 01 CD 5D 25 32 02 41 FE 2A CA FD 21 21 01 41 ...]%2.A.*..!!.A
21B0 36 00 CD 5D 25 F5 3A 01 41 3C 32 01 41 4F 06 00 6..]%.:.A<2.AO..
-
```

Figure 2

```
D2100
2100 0F 0F 0F E6 60 C6 0F 2A 05 1A 4F 06 00 09 7E C6 ....`..*..O...~.
2110 07 0F 0F 0F E6 1F 4F E1 09 E5 0E 12 C3 F8 20 CD ......O....... .
2120 5E 1A E1 AF 32 1A 1A C9 4E 4F 20 49 4E 54 45 52 ^...2...NO INTER
2130 4D 45 44 49 41 54 45 20 4C 41 4E 47 55 41 47 45 MEDIATE LANGUAGE
2140 20 46 49 4C 45 20 24 04 50 6C 65 61 73 65 20 77  FILE $.Please w
2150 61 69 74 2E 2E 2E 24 24 24 24 24 24 21 00 01 22 24 ait...$$$$$$!..."$
2160 41 21 50 00 22 26 41 3A 45 41 1F DA 86 21 CD E6 A!P."&A:EA...!..
2170 17 01 47 21 CD 1A 27 0E 0D CD C4 26 0E 0A CD C4 ..G!..'....&....
2180 26 21 EF 01 36 00 CD CC 24 21 0E 43 22 FC 40 CD &!..6...$!.C".@.
2190 5D 25 4F 3E 02 B9 D2 A2 21 01 56 49 CD 1D 28 CD ]%O>....!.VI..(.
21A0 0D 01 CD 5D 25 32 02 41 FE 2A CA FD 21 21 01 41 ...]%2.A.*..!!.A
21B0 36 00 CD 5D 25 F5 3A 01 41 3C 32 01 41 4F 06 00 6..]%.:.A<2.AO..
-
```

Figure 3

```
-S2147
2147 43 04
2148 52 50
2149 55 6C
214A 4E 65
214B 20 61
214C 56 73
214D 45 65
214E 52 20
214F 20 77
2150 32 61
2151 2E 69
2152 30 74
2153 35 2E
2154 24 2E
2155 24 2E
2156 24 X
?
-
```

Figure 4

```
 1: REM     ****************************************\
 2:         *     CBASIC LINE EDITOR.                *\
 3:         * THIS PROGRAM DEMONSTRATES THE CBASIC   *\
 4:         * SINGLE CHARACTER OUTPUT ROUTINE        *\
 5:         *                                        *\
 6:         *       WRITTEN JULY, 1980               *\
 7:         *       ANDY & BEN GALEWSKY              *\
 8:         ****************************************
 9: REM     FIRST DEFINE THE PRINT SINGLE CHARACTER FUNCTION
10:
11:         DEF FN.PRN(DISP$)
12:                 POKE 130H,ASC(DISP$)            REM CHARACTER TO BE PRINTED
13:                                                 REM IS PASSED IN 130 HEX
14:                 CALL 120H                       REM CALL ROUTINE
15:                 RETURN
16:         FEND
17: CLR$=CHR$(4)    REM SCREEN CLEAR CHARACTER FOR VECTOR MINDLESS TERMINAL
18:                 REM CHANGE TO SCREEN CLEAR ON YOUR TERMINAL
19: REM     START OF PROGRAM
20: 5       PRINT "ENTER LINE TO EDIT "
21:         INPUT " "; LINE EDIT$
22:         PRINT CLR$
23:         PRINT EDIT$     REM DISPLAY LINE AT TOP OF SCREEN
24:         POINTER%=1      REM CHARACTER POINTER
25:         DUM=FN.PRN("^") REM PLACE MARKER ON SCREEN
26: 10      INKEY%=CONCHAR% REM GET ONE KEYBOARD CHARACTER
27:         IF INKEY%=32 THEN 20    REM SPACE BAR
28:         IF INKEY%=ASC("D") THEN 30      REM DELETE CHARACTER
29:         IF INKEY%=ASC("C") THEN 40      REM CHANGE
30:         IF INKEY%=13 THEN 50            REM RETURN
31:         GOTO 10
32: 20      POINTER%=POINTER%+1             REM INCREMENT POINTER
33:         DUM=FN.PRN(CHR$(8))             REM MOVE CURSOR BACK
34:         DUM=FN.PRN(CHR$(8))
35:         DUM=FN.PRN(" ")                REM ERASE OLD MARKER AND GO FORWARD
36:         DUM=FN.PRN("^")                 REM PRINT NEW MARKER
37:         GOTO 10
38:
39: REM     DELETE CHARACTER
40:
41: 30      LEF$=LEFT$(EDIT$,POINTER%-1)    REM GET LEFT OF DELETION
42:         RIG$=MID$(EDIT$,POINTER%+1,LEN(EDIT$))  REM GET RIGHT
43:         PRINT CLR$
44:         EDIT$=LEF$+RIG$                         REM REBUILD STRING
45:         PRINT EDIT$                             REM PRINT IT
46:         POINTER%=POINTER%-1                     REM DECREMENT CHARACTER POI
47:         FOR MOV%=1 TO POINTER%-1                REM REPOSITION CURSOR
48:         DUM=FN.PRN(" ")
49:         NEXT MOV%
50:         DUM=FN.PRN("^")                         REM PRINT MARKER
51:         GOTO 10
52:
53: REM     CHANGE CHARACTER
54:
55: 40      REPL$=CHR$(CONCHAR%)                    REM GET CHANGE
56:         LEF$=LEFT$(EDIT$,POINTER%-1)            REM LEFT PART
57:         RIG$=MID$(EDIT$,POINTER%+1,LEN(EDIT$))  REM RIGHT PART
58:         PRINT CLR$
59:         EDIT$=LEF$+REPL$+RIG$                   REM REBUILD STRING
60:         PRINT EDIT$                             REM PRINT IT
61:         FOR MOV%=1 TO POINTER%-1                REM REPOSITION POINTER
62:         DUM=FN.PRN(" ")
63:         NEXT MOV%
64:         DUM=FN.PRN("^")                         REM DISPLAY MARKER
65:         GOTO 10
66:
67: REM     GET NEW LINE TO EDIT
68:
69: 50      PRINT CLR$
70:         GOTO 5
```

Using CP/M's Undocumented "Autoload" Feature

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 "pump" what we just entered...
                          .--- 15 characters in MBASIC STAR?REK
                          |
                          |  .--- Start of MBASIC STAR?REK
                          |  |
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 4B 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

The Godbout Dual Processor Board and CP/M-86

Bruce Ratoff

Well, by now it seems like you've always had that Z80A running at "4 Meg," and the full 64K of high-speed RAM you got to go with it has collected a nice layer of dust since you haven't changed a board in months. Your bank account is finally recuperating from the purchase of that double sided double density disk system you bought a few months back. Right about now, you're congratulating yourself on finally putting together a state-of-the-art system. Guess again! The 16-bit micros have finally come alive, with enough off-the-shelf hardware and software available to make assembling a 16-bit S-100 system a reasonable project for an experienced microcomputerist.

For the past few months I have had the opportunity to install and use Godbout's 8085/8088 Dual Processor Board with Digital Research's new 8086 implementation of the CP/M operating system. The hardware and software were received in their standard, unconfigured form. I was thus able to experience the installation of this new processor and operating system on an existing system. Through this report, I hope to convey to you my impression of these two powerful and exciting new tools.

A Quick Look

The Godbout Dual processor, as the name implies, contains an 8085 microprocessor for the execution of existing 8080-family software, along with an 8088 microprocessor for the execution of the newer 8086-family software. The system powers up with the 8085 active. By means of a software command, the user may then switch back and forth between it and the 8088. This is accomplished by an input command to an I/O port, whose address is switch selectable on the card. An output to the same I/O port sets the value of extended address lines A16 through A23, allowing the 8085 to overcome its normal 64 kilobyte addressing limits and access all 16 megabytes defined by the IEEE-696 standard. Only the upper four bits of this port are used when the 8088 is active, since this processor has built-in addressing for 1 megabyte.

Bruce R. Ratoff, 26 Broad Street, Cranford, NJ 07016.

The 8085 chip is basically an enhanced 8080, which eliminates the clock generator chip and negative power supply required for an 8080 system. It also practically eliminates the need for an interrupt controller chip in systems requiring interrupts, since input pins and vectoring hardware are provided on the processor for four new interrupts, in addition to the non-vectored interrupt carried over from the original 8080. One of these, the Non-Maskable Interrupt, is brought out to the newly-defined NMI pin of the S-100 bus. The remaining three new interrupts, which are maskable in software, may be jumpered to any of the eight S-100 vectored interrupt pins. These three new interrupts are referred to as RST 5.5, RST 6.5 and RST 7.5, since they generate calls to addresses 4 bytes above the original 8080's RST 5, RST 6 and RST 7 instructions. The 8085 instruction set is identical to that of the 8080, with the addition of two instructions to enable and disable the three new maskable interrupts. It is important to note that the additional Z80 instruction set is *not* implemented. A premium version of the 8085 is used on the Godbout board, allowing operation with a 5 MHz clock rate. A switch is provided to drop the 8085's speed to 2 MHz, to accommodate older (and slower) memory boards.

The 8088 contains pipeline logic which will fetch up to the next four memory bytes while the current instruction is being decoded and executed.

The 8088 microprocessor chip represents Intel's recognition of the large number of microprocessor users who would like to upgrade to a 16-bit microprocessor without having to convert all their 8-bit bus hardware and peripherals. The result is an 8086 processor which has been internally modified to convert each 16-bit memory or port access into two sequential 8-bit accesses.

The 8088 contains pipeline logic which will fetch up to the next four memory bytes while the current instruction is being decoded and executed. Internal operations may therefore proceed at full 16-bit speed, resulting in an overall execution speed almost equivalent to that obtainable on a true 16-bit bus. The bus timing for memory accesses was also made somewhat more liberal, with the result that an 8088 operating a 5 MHz (as on Godbout's board) will work with most memory designed for 2 or 3 MHz 8-bit systems, without the need to add wait states. Godbout apparently found this to be true, since no means is provided to slow the 8088's 5 MHz clock.

CP/M-86 is Digital Research's first venture into the 16-bit micro software market. It implements the same basic file structure, utilities and commands as the current version (2.2) of 8-bit CP/M. Disks written by the two systems are fully interchangeable, as long as the same disk definitions are used in the 8- and 16-bit BIOSes. 8086 equivalents of all the standard CP/M utilities such as ASM, PIP, ED and DDT are provided. Those programs necessary to configure the system (such as the 8086 assembler) are also provided in 8080-executable form. This should allow the use of an existing CP/M-80 system to develop and install a CP/M-86 BIOS. All the CP/M-80 version 2.2 BDOS calls are present and use the same function numbers, easing the task of converting existing programs. New BDOS functions have been added to provide controlled access to the 8086 memory management features.

Testing

Two system configurations were used to test the hardware and software. The main one consisted of a non-front panel enclosure, containing a Vector motherboard, an Imsai SIO2-2 serial interface, an iCom 3712 8-inch single density diskette subsystem, and 64K of various brands and speeds of static RAM. It should be pointed out that some of the memory was already known not to operate with a 4 MHz Z80A. The iCom disk system seemed like a good choice for a first attempt at bringing up CP/M-86, since it used a buffered controller and simple parallel interface with no wait state insertion or special timing requirements. The second test system was an Imsai I8080 front-panel type system, containing the original Imsai motherboard, two SSM I04 I/O boards for serial I/O, 64K of fast static RAM and an Industrial Micro Systems 400 diskette controller. This configuration allowed me to test the Godbout board's operation in the potentially troublesome areas of DMA (on the IMS controller) and front panel operation. Time did not permit installing CP/M-86 on the second system, so the software part of this review is based on operation with the iCom disks only.

Hardware Evaluation

The Godbout board gives a very good first impression as it comes out of its shipping carton. The layout appears clean and open, in spite of the fact that the board contains over 40 IC's. The two five volt regulators sit on the left side (where the vents are on most S-100 cabinets), balanced by the two 40-pin microprocessor IC's on the right. In the upper right corner is a 16-pin DIP socket for the optional connection to a front panel. Card ejectors are provided in the upper corners of the board (I wish more manufacturers would provide these, as they prevent skinned knuckles when changing cards in a tight motherboard). The board is solder-masked on both sides, and appears to have been wave-soldered. The silkscreened legends on the component side of the board identify each IC by both its sequential number in the schematic, as well as its generic type number (7400, 8085, etc.). Each option switch (and there are many) has its function clearly marked. One minor annoyance is the absence of metal "fingers" on the unused S-100 connector pins. The high cost of gold plating has caused a lot of manufacturers to omit these, but the result is that the motherboard sockets become dirty sooner, and the user is prevented from making any hardware modifications that might have required the additional pins.

While there are a great many option switches to be set on this board, most are more or less self-explanatory. In either case, the manual explains them in detail and shows the most common initial setup. A large red toggle switch near the upper right corner of the board selects between 2 MHz and 5 MHz operation of the 8085 processor (the 8088 is fixed at 5 MHz). There are three sets of 8 DIP switches. The one in the bottom row selects the I/O port number used to control the processor. An output to this port sets the extended address lines. An input returns meaningless data, but causes control to switch from the current processor to the other one. I set this to the recommended value of OFD hex. The middle set of switches sets the address for the power-on-jump logic to any 256-byte boundary. I used the address of the disk boot PROM in each of my systems. The last set of switches, located near the top of the board, control miscellaneous options. These include: whether to disable the extended address lines during DMA, whether to clear the extended address lines (to all 0's) at each reset, whether to insert wait states in all I/O operations, whether to reset each processor every time it becomes active or let it continue from where it was, whether to do a jump on reset, whether to do a power-on-jump, and whether to generate the S-100 MWRITE signal. I selected power-on-jump and jump-on-reset in both systems. MWRITE generation was required only in the non-front panel system, since the front panel of my Imsai does its own generation of this signal. I selected the "continue" mode of operation for both processors. However, I did install an additional jumper, described in an addendum to the manual, which allowed the bus reset button to affect both processors, rather than just the 8085. I discovered through experimentation that the I/O wait option was only necessary when operating the 8085 at 5 MHz. All my I/O devices seemed to work fine without wait states when the 8088 was in control.

I was quite pleased with the operation of the board in both systems. Once the correct options were set up, the board performed flawlessly. I have run just about every popular CP/M-based language and package on the 8085 section of the board without any problems. Once potential "catch" concerns operation of the board with DMA devices: due to the manner in which the processor changeover is accomplished, one cannot use the "reset or changeover" option when DMA devices are present, since the DMA is

Chapter VI
CP/M — 86

seen as a processor changeover and causes a reset to occur. This should pose no problem in running CP/M-86, since the reset feature is not required.

By now I'm sure some of you are saying "but why couldn't they have used a Z80 instead of the 8085?" The reason is simple—there is a great similarity between the timing of the 8085 and 8088 processors. Intel did this to make it easy for their industrial users to adapt existing 8085 designs to the 8088. In the case of the Godbout board, it allows the two processors to share most of the S-100 bus interface logic. Since the timing of the Z80 is vastly different, it probably would have necessitated two totally separate interface circuits, which would not have fit on a single S-100 card. There may be some hope, however. National Semiconductor makes a processor called the NSC800, which they claim has the Z80 instruction set, but timing similar to an 8085. Unfortunately, the NSC800 and the 8085 are not pin-compatible, so some wiring changes would be necessary. Also, the chip seems to be in relatively short supply. Maybe someone at Godbout should be looking into the use of this chip in some future revision to the board (are you listening, Mr. G?).

The first thing that struck me about CP/M-86 was the remarkable degree of similarity to CP/M-80 in both the user and system levels of interface.

There is really only one feature of this board that in my opinion does not live up to expectations. That is the "powerful memory management" alluded to in the company's advertising. What is actually provided on the board would be more accurately called "centralized bank switching." There is a single parallel port with its outputs connected to S-100 address lines 16 through 23 (when the 8085 is in control) or 20 through 23 (when the 8088 is in control). The trouble with this simple scheme is that the output instruction which sets the extended address lines must be executed from a memory card that doesn't recognize the extended address. Otherwise, the program would be knocking its own memory out from under itself! This is not much of a problem when running 8-bit software such as MP/M, which requires some non-banked memory for parts of the operating system anyway. It is also not a serious problem for the 8088, since the CPU directly addresses a megabyte before bank switching is required. The hassle comes when the two processors are used together, if the 8085 needs to access memory above the first 64K to perform some task for the 8088. An example would be the setting up of the 8088's reset vector (at address OFFFFO hex) prior to switching control from 8085 to 8088. The non-extended memory required to perform this operation would require a gap the size of the non-extended card to be left in each 64K of the 8088's one megabyte space, reducing the maximum size of each 8088 memory segment by the size of the non-extended card. A possible solution to the specific problem of starting up the 8088 is to use a PROM monitor in the extended address space. Alternatively, the extended PROM could simply contain a jump instruction to somewhere in the first 64K, making extended references by the 8085 unnecessary. In any event, I would hope that future processors adopt some true form of address translation or mapping so that practical use may be made of the full addressing capabilities of the S-100 bus.

Software Evaluation

The first thing that struck me about CP/M-86 was the remarkable degree of similarity to CP/M-80 in both the user and system level of interface. This consistency helped me to immediately feel at home, in spite of the fact that I was on a brand new processor and operating system. The software comes on two 8 inch single density floppies. A looseleaf binder contains copies of the *CP/M 2.2 Users Guide*, the *ED Users Manual* and *An Introduction to CP/M Features and Facilities*, all of which are the same manuals supplied with the CP/M-80. Three new manuals provided are the *CP/M-86 System Reference Guide*, the *CP/M-86 Assembler Users Guide*, and the *DDT-86 Users Guide*. *The System Reference* appears to be the equivalent of both the "Interface Guide" and "Alteration Guide" found in the CP/M-80 documentation package. These manuals seem to be best organized for looking things up rather than reading straight through. All the necessary information is presented in a well organized manner, with several example programs provided both in the appendices and on the release diskettes. There is a great deal of information presented, but it does all fall into place quickly.

CP/M-86 is larger than CP/M-80, and therefore does not fit on the two system tracks of a standard diskette. Instead, it sits in a file called CPM.SYS. An abbreviated version of the system occupies the system tracks, and is used to load the system file during boot-up. Unlike CP/M-80, the system is not reloaded every time a program exits. Control-C issued to a running program simply causes a return to the CCP prompt. Control-C to the CCP causes the disks to be re-logged in. CP/M-86 takes advantage of the inherent relocatability of 8086 object code. The system may be loaded anywhere in memory without the need for a MOVCPM-like program. The normal procedure is to boot the system into address 00400 hex, just above the 8086 interrupt vector area. This leaves memory from about 02A00 and up free for loading programs.

In CP/M-86, the familiar .COM file type for executable code has been replaced by a new .CMD file type. Besides denoting the presence of 8086 object code rather than 8080, the .CMD file has a header record that describes the program's space requirements for code, data and stack space. This results in much more compact program storage on disk. A new utility called GENCMD is used to create .CMD files from the extended hex (.H86) files produced by the assembler. This replaces the LOAD program found in a CP/M-80 system. The executable files thus produced may use one of three memory configurations: the "8080 model," in which code and data are given a single memory area of up to 64K, the "small model," where two separate areas of up to 64K each are allotted for code and data, or the "compact

model," in which up to eight separate memory areas of up to 64K each may be allocated for code and data. The necessary configuration is determined automatically by the system from the information contained in the .CMD header record.

The interface between a program and the system has been modified slightly. The page 0 BIOS and BDOS vectors of CP/M-80 have been done away with. Instead, the 8086 software interrupt instruction is used to perform BDOS calls. Since there is no more "warm boot vector" at location 0 for performing direct BIOS calls, a new BDOS function has been added for direct access to all the BIOS routines. The IOBYTE has been moved from location 0003 into the BIOS, with two new calls added to read and set it. Instead of an absolute page 0, the first page of the program's data segment is used by the system to pass the amount of available memory, the default FCB's, and the default I/O buffer. When the "8080 model" configuration is used, this will result in a setup nearly identical to CP/M-80. Due to the absence of a warm boot vector, program termination via "jmp 0" is no longer possible. The program must do a BDOS function 0, or an 8086 "return far" instruction to exit back to the operating system.

CP/M-86 contains added BDOS functions to handle the 8086's memory segmentation features. An added BIOS function allows you to specify a table of up to eight non-contiguous areas of memory for programs and data. This allows you to bypass any ROM or other dead blocks in your system. CP/M-86 will then further divide the areas you specify if necessary to provide a total of up to eight separate memory segments. New BDOS calls are provided to allow a program to request additional memory, and to request another program to be loaded. This means that programs may call each other in nested fashion up to eight levels deep.

The CP/M built-in commands remain just about the same as before. DIR, ERA, REN, TYPE and USER operate identically to CP/M-80. The SAVE command has been done away with, however, due to the confusion that it would cause in a segmented memory environment (how would you know which area to save?). Instead of SAVE, a Write command has been added to DDT for saving patched object files. The other noticable difference at the keyboard is that control-P is no longer cancelled. when a program terminates or control-C is typed. It will remain in effect indefinitely, until another control-P is typed. This greatly improves your ability to get hardcopy of your console output.

I found installing my first CP/M-86 to be much easier than what I recall of my first few attempts with CP/M-80 back in the days of version 1.3. I simply took a listing of my current CP/M-80 BIOS, hand-translated the disk and console portions into 8086 mnemonics, and edited them into the CP/M-86 BIOS skeleton provided on one of the release diskettes. I then used the thoughtfully-provided ASM86.COM to assemble the new BIOS on the 8085 and CP/M-80. Because of the relocatability of 8086 code, there are no equates in the BIOS for memory size (although there is the aforementioned table of available memory areas), and the whole mess of calculating load offsets for DDT has been eliminated. One simply used PIP to concatenate the provided CPM.H86, which contains the CCP and BDOS, with your just-assembled CBIOS.H86. GENCMD.COM, an 8080-executable version of the CP/M-86 program loader, is used to turn the combined hex file into an 8086 object file. At this point came the big question: "Now that I've got it, how do I boot this thing?" This is where having both processors on one board really paid off. I simply wrote a short preamble for CPM.SYS in 8085 code, which set the 8086 reset vector to jump to the 8086 BIOS and then switched processors. Voila! A CP/M-86 system that executes as a CP/M-80 .COM file. As a finishing touch, I would later make this the embedded command in my CP/M-80 system, so that I could appear to boot straight into CP/M-86.

With the details of starting up the system worked out, it was time to begin testing. I keyed in the command "CPM86" (I had saved the 8086 system with the 8085 preamble as CPM86.COM) and waited. In a few seconds, I was quite tickled to see the message:

CP/M-86 Version 1.0
System Generated 03/15/81

and then...

Nothing! The system had printed the signon and then hung up somewhere. Well, let's see. Since the signon printed, the console routines must be working, so the problem must be somewhere in the disk logic, when it goes to log in drive A. The code looks OK, so what am I missing? Wait a minute! Let's have a look at that iCom schematic. Just as I suspected, it's decoding the port number from the upper address bus. This is a common problem on older S-100 boards, where the layout designer took advantage of the fact that the 8080 duplicates the I/O port number on address lines 8 through 15. Most S-100 Z80 cards have extra logic to perform this function, so there's no problem there, but what do you do on a processor like the 8088 that allows port numbers greater than 255? (In fact, the 8088 uses 16 address bits for port numbers, allowing 64K of I/O ports.) Well, back into CP/M-80, and find a way to make it work. Aha! I can write the 8086 code using 16-bit port numbers that have the same lower and upper byte. That should keep all the old boards happy. The only drawback is that to get the 16-bit port numbers requires loading the CX register with the port number before each I/O instruction, since that's the only means provided on the 8088 for accessing the higher port numbers. Anyway, a few quick edits, reassemble and try it again. This time, the system signs on, and I get the familiar "A >" prompt. Fantastic! I type "DIR", and the system responds (a bit more rapidly than CP/M-80, I believe) with a proper directory listing. TYPE also seems to be doing its thing. OK, I know the disk read logic must be working, so the next step is to try to write a file. In this case, I tried to PIP something into another file. No go. After I reboot the system, I can see the new name that was created in the directory, so it must be almost working. Examination of the disk write code showed that I had forgotten to pop a register, so I fixed that and tried again. Still just as bad! At this point, I got an object lesson on the effect of the segment registers. I had changed the data segment register in order to obtain the data to be written from the calling program's data segment. Since I forgot to set it back to my own

data segment, all further references to my BIOS variables were coming from somewhere south of Lower Slobbovia! Another well-placed push/pop pair and disk writes started behaving themselves. *There I finally was, with a real live and working CP/M-86 system!* I then used the working CP/M-86 system to further enhance the CBIOS with a handshaking list driver for my Diablo printer, and various other minor bells and whistles. Once I had set up CPM86.COM for auto-execute from CP/M-80, I was ready to log some program development time.

The difference in speed between the .COM and .CMD versions of ASM86 was immediately noticeable, although not quite as great as I would have expected. GENCMD was drastically improved, with the .COM version seeming to take forever, while the .CMD was about as fast as the CP/M-80 LOAD program. ED, PIP and STAT all seemed slightly faster, while SUBMIT seemed about the same. One can reasonably assume that compute-bound programs will benefit the most, especially if they are partially rewritten to take advantage of the 8088's added instruction set. Disk-bound programs are of course limited by the disk transfer rate and won't show much improvement.

As a final example, I converted my Super Directory program from the SIG/M library into 8086 code. This program contained many opportunities to take advantage of the 8086, since it contains a character-string sort routine and a large number of 16-bit computations. I recoded the sort routine to use the 8086 string-compare routine, thereby eliminating about twenty lines of code. I changed the decimal output routine to use the hardware divide instruction, shortening that code. The ability to store constants directly into memory, as well as the ability to increment and decrement memory directly, without the use of a pointer register, were very useful throughout the program. The index registers and multiple bit shifts were also put to good use. The end result of my work on CBIOS and SD appears at the end of this article, and will be available on a SIG/M library diskette at some later date as part of a collection of 8086 programs.

The one program which requires a bit of getting used to is the 8086 assembler, ASM86. As was stated in the manuals, this assembler is mostly faithful to the Intel standard in mnemonics and basic design. The main area of deviation is that inter-segment jumps, calls and returns have unique mnemonics rather than being detected automatically. The tricky part of the Intel standard is that the code generated when a particular identifier is used depends on how that identifier was defined. If it was defined by an EQU, for example, it is treated as a numeric literal and generates an immediate-mode instruction. The label of a DB instruction causes an 8-bit instruction to be generated wherever it is referenced, while DW's cause 16-bit instructions to be generated. Code labels cannot be used in data-reference instructions, and will produce an error message from the assembler. One "feature" which does not seem to be mentioned in the manuals is that code labels must be followed by a colon (:), while data labels *must not* be, and will cause error messages at every reference to that label. While this is no problem when writing new code, it caused a bit of head-scratching at first when converting existing programs. Also, for some reason the "jump carry" (jc) and "jump not carry" (jnc) opcodes seem to be missing from the assembler. Once again, this is only a problem with existing code, since the synonyms "jump below" (jb) and "jump above or equal" (jae) are present and work properly.

The dual processor board makes it possible to step up to 16 bits without sacrificing any existing hardware, or having to swap CPU cards to run 8 bit software.

Conclusions

In spite of some of the minor problems mentioned here, both the hardware and software tested appear to be solid, reliable tools which may be had at a very reasonable cost. The dual board makes it possible to step up to 16 bits without sacrificing any existing hardware, or having to swap CPU cards to run 8-bit software. Likewise, CP/M-86 allows a smooth upgrade to 16-bit programming without the need to learn a totally new operating environment. Given the similarity between the 8086/88 and 8080/Z80 architectures, combined with the familiarity of CP/M, most programmers and their software should make the transition with ease. Digital Research is to be congratulated for once again providing a standard-setting product that will provide a consolidated market for the software of the 1980's.

With these products and the others which will now surely follow, 16-bit computing has finally arrived!

```
                        title   'Customized Basic I/O System'

                ;*****************************************
                ;*                                       *
                ;* This Customized BIOS adapts CP/M-86 to *
                ;* the following hardware configuration  *
                ;*      Processor:  8085/8088 Dual Processor *
                ;*      Brand:  CompuPro (Godbout)       *
                ;*      Controller:  iCom 3712           *
                ;*                                       *
                ;*                                       *
                ;*      Programmer:  Bruce R. Ratoff     *
                ;*      Revisions :  04/30/81 20:40      *
                ;*                                       *
                ;*****************************************

FFFF            true            equ -1
0000            false           equ not true
000D            cr              equ 0dh ;carriage return
000A            lf              equ 0ah ;line feed

                ;*****************************************
                ;*                                       *
                ;* Loader bios is true if assembling the *
                ;* LOADER BIOS, otherwise BIOS is for the *
                ;* CPM.SYS file.                         *
                ;*                                       *
                ;*****************************************

0000            loader bios     equ false
00E0            bdos int        equ 224 ;reserved BDOS interrupt

                ;*****************************************
                ;*                                       *
                ;*          I/O Port Assignments         *
                ;*                                       *
                ;*****************************************
                ;
                ;Diskette interface (iCom 3712)
                ;Note: Port numbers are "doubled up" because iCom card
                ;       counts on 8080 "address mirror" effect.
C0C0            datai   equ     0c0c0h  ;data/status input port
C1C1            datao   equ     0c1c1h  ;data output port
C0C0            cntrl   equ     0c0c0h  ;command output port
                ;
                ;Console interface (IMSAI SIO2-2 port 1)
0003            cstat   equ     3       ;status
0002            cdata   equ     2       ;data
0002            cimsk   equ     2       ;input ready mask
0001            comsk   equ     1       ;output ready mask
                ;
                ;Printer interface (IMSAI SIO2-2 port 2)
0005            lstat   equ     5       ;status
0004            ldata   equ     4       ;data
0001            lomsk   equ     1       ;output ready mask
0002            limsk   equ     2       ;input ready mask

                        IF      not loader bios
                ;-----------------------------------------
                ;|                                       |
2500            bios code       equ 2500h
0000            ccp offset      equ 0000h
0806            bdos ofst       equ 0806h ;BDOS entry point
                ;|                                       |
                ;-----------------------------------------
                        ENDIF   ;not loader bios

                        IF      loader bios
                ;-----------------------------------------
                ;|                                       |
                bios code       equ 1200h ;start of LDBIOS
                ccp offset      equ 0003h ;base of CPMLOADER
                bdos ofst       equ 0406h ;stripped BDOS entry
                ;|                                       |
                ;-----------------------------------------
                        ENDIF   ;loader bios

                        cseg

                        org     ccpoffset
                ccp:
                        org     bios code

                ;*****************************************
                ;*                                       *
                ;* BIOS Jump Vector for Individual Routines *
                ;*                                       *
                ;*****************************************

2500 E9 3C 00           jmp INIT        ;Enter from BOOT ROM or LOADER
2503 E9 85 00           jmp WBOOT       ;Arrive here from BDOS call 0
2506 E9 C8 00           jmp CONST       ;return console keyboard status
2509 E9 CE 00           jmp CONIN       ;return console keyboard char
250C E9 D5 00           jmp CONOUT      ;write char to console device
250F E9 DD 00           jmp LISTOUT     ;write character to list device
2512 E9 20 01           jmp PUNCH       ;write character to punch device
2515 E9 1E 01           jmp READER      ;return char from reader device
2518 E9 54 01           jmp HOME        ;move to trk 00 on cur sel drive
251B E9 32 01           jmp SELDSK      ;select disk for next rd/write
251E E9 51 01           jmp SETTRK      ;set track for next rd/write
2521 E9 58 01           jmp SETSEC      ;set sector for next rd/write
2524 E9 61 01           jmp SETDMA      ;set offset for user buff (DMA)
2527 E9 6C 01           jmp READ        ;read a 128 byte sector
252A E9 AD 01           jmp WRITE       ;write a 128 byte sector
252D E9 DC 00           jmp LISTST      ;return list status
2530 E9 4E 01           jmp SECTRAN     ;xlate logical->physical sector
2533 E9 57 01           jmp SETDMAB     ;set seg base for buff (DMA)
2536 E9 59 01           jmp GETSEGT     ;return offset of Mem Desc Table
2539 E9 FD 00           jmp GETIOBF     ;return I/O map byte (IOBYTE)
253C E9 FE 00           jmp SETIOBF     ;set I/O map byte (IOBYTE)

                ;*****************************************
                ;*                                       *
                ;* INIT Entry Point, Differs for LDBIOS and *
                ;* BIOS, according to "Loader Bios" value *
                ;*                                       *
                ;*****************************************

                INIT:   ;print signon message and initialize hardware
253F 8C C8              mov ax,cs       ;we entered with a JMPF so use
2541 8E D0              mov ss,ax       ;CS: as the initial value of SS:,
2543 8E D8              mov ds,ax       ;DS:,
2545 8E C0              mov es,ax       ;and ES:
                        ;use local stack during initialization
2547 BC BE 29           mov sp,offset stkbase
254A FC                 cld             ;set forward direction

                        IF      not loader bios
                ;-----------------------------------------
                ;|                                       |
                        ; This is a BIOS for the CPM.SYS file.
                        ; Setup all interrupt vectors in low
                        ; memory to address trap

254B 1E                 push ds         ;save the DS register
254C C6 06 8C 27 00     mov IOBYTE,0    ;clear IOBYTE
2551 B8 00 00           mov ax,0
2554 8E D8              mov ds,ax
2556 8E C0              mov es,ax       ;set ES and DS to zero
                        ;setup interrupt 0 to address trap routine
2558 C7 06 00 00 9A 25  mov int0 offset,offset int trap
255E 8C 0E 02 00        mov int0 segment,CS
2562 BF 04 00           mov di,4
2565 BE 00 00           mov si,0        ;then propagate
2568 B9 FE 01           mov cx,510      ;trap vector to
256B F3 A5              rep movs ax,ax  ;all 256 interrupts
                        ;BDOS offset to proper interrupt
256D C7 06 80 03 06 0B  mov bdos offset,bdos ofst
2573 C7 06 00 00 8E 25  mov int0 offset,offset int0 trap
2579 C7 06 10 00 94 25  mov int4 offset,offset int4 trap
257F 1F                 pop ds          ;restore the DS register

                ;       (additional CP/M-86 initialization)
                ;|                                       |
                ;-----------------------------------------
                        ENDIF   ;not loader bios
```

```
                        IF      loader bios
                ;---------------------------------------
                ;|                                     |
                        ;This is a BIOS for the LOADER
                        push ds         ;save data segment
                        mov ax,0
                        mov ds,ax       ;point to segment zero
                        ;BDOS interrupt offset
                        mov bdos offset,bdos ofst
                        mov bdos segment,CS ;bdos interrupt segment
                ;       (additional LOADER initialization)
                        pop ds          ;restore data segment
                ;|                                     |
                ;---------------------------------------
                        ENDIF   ;loader bios

2580 BB 97 27           mov bx,offset signon
2583 E8 BC 00           call pmsg       ;print signon message
2586 B1 00              mov cl,0        ;default to dr A: on coldstart
2588 E9 75 DA           jmp ccp         ;jump to cold start entry of CCP

2588 E9 78 DA   WBOOT:  jmp ccp+6       ;direct entry to CCP at command level

                        IF      not loader bios
                ;---------------------------------------
                ;|                                     |
                int0 trap:
258E FA                 cli
258F BB F4 27           mov bx,offset int0 trp
2592 EB 0A              jmps int halt
                int4 trap:
2594 FA                 cli
2595 BB 0B 28           mov bx,offset int4 trp
2598 EB 04              jmps int halt
                int trap:
259A FA                 cli             ;block interrupts
259B BB DA 27           mov bx,offset int trp
                int halt:
259E 8C C8              mov ax,cs
25A0 8E D8              mov ds,ax       ;get our data segment
25A2 E8 9D 00           call pmsg
25A5 5B                 pop bx          ;get offset
25A6 58                 pop ax          ;print segment
25A7 53                 push bx         ;save offset
25A8 E8 CA 00           call PHEX
25AB B1 3A              mov cl,':'      ;colon
25AD E8 34 00           call CONOUT
25B0 58                 pop ax          ;print offset
25B1 E8 01 00           call PHEX
25B4 F4                 hlt             ;hardstop

                PHEX:
25B5 50                 push ax
25B6 8A C4              mov al,ah
25B8 E8 01 00           call PHXB       ;print upper byte
25BB 58                 pop ax          ;restore to print lower byte
                PHXB:
25BC 50                 push ax         ;save byte
25BD B1 04              mov cl,4        ;get high nibble
25BF D2 E8              shr al,cl       ;into low bits
25C1 E8 03 00           call PHXD       ;print digit
25C4 58                 pop ax          ;restore byte
25C5 24 0F              and al,0fh      ;isolate low nibble
                PHXD:
25C7 04 90              add al,90h      ;first half of conversion trick
25C9 27                 daa
25CA 14 40              adc al,40h      ;second half of same
25CC 27                 daa
25CD 8A C8              mov cl,al       ;now print digit
25CF EB 13              jmps CONOUT
                ;|                                     |
                ;---------------------------------------
                        ENDIF   ;not loader bios

                ;**************************************
                ;*                                    *
                ;*   CP/M Character I/O Interface Routines   *
                ;*                                    *
                ;**************************************

25D1 E4 03              CONST:  ;console status
25D3 24 02                      in      al,cstat        ;get status byte
25D5 74 02                      and     al,cimsk        ;check input mask
25D7 0C FF                      jz      const1          ;not ready yet...return al=0, ZF=1
25D9 C3                         or      al,0ffh         ;ready...return al=0FFh, ZF=0
                        CONST1:
                                ret

25DA E8 F4 FF           CONIN:                  ;console input
25DD 74 FB                      call CONST
25DF E4 02                      jz CONIN        ;wait for RDA
25E1 24 7F                      in      al,cdata;get byte
25E3 C3                         and     al,7fh  ;strip parity
                                ret

25E4 E4 03              CONOUT:         ;console output
25E6 A8 01                      in      al,cstat        ;get status
25E8 74 FA                      test    al,comsk        ;check output bits
25EA 8A C1                      jz      conout          ;loop till ready
25EC E6 02                      mov     al,cl           ;setup
25EE C3                         out     cdata,al        ;send character
                                ret             ;then return data

25EF E8 1A 00           LISTOUT:                ;list device output
25F2 74 FB                      call    LISTST  ;get output status
25F4 8A C1                      jz      LISTOUT ;wait for TBE
25F6 E6 04                      mov     al,cl   ;setup
25F8 E4 05                      out     ldata,al        ;send char
25FA 24 02                      in      al,lstat        ;check for handshake received
25FC 74 0D                      and     al,linsk
25FE E4 04                      jz      LISTOUT2        ;no handshake...exit
2600 24 7F                      in      al,ldata        ;get handshake char
2602 3C 13                      and     al,7fh          ;strip parity
2604 75 05                      cmp     al,'S'-40h      ;XOFF?
2606 C6 06 88 27 FF             jnz     LISTOUT2        ;nope
                                mov     lstactive,0ffh  ;set list active flag
                        LISTOUT2:
2608 C3                         ret

260C E4 05              LISTST:                 ;poll list status
260E 24 01                      in      al,lstat        ;get status byte
2610 74 20                      and     al,lomsk        ;test output bits
2612 A0 88 27                   jz      LISTST1         ;not ready...exit with al=0, zf=1
2615 F6 D0                      mov     al,lstactive    ;line ready...waiting for XON?
2617 84 C0                      not     al
2619 75 17                      test    al,al
261B E4 05                      jnz     LISTST1         ;not waiting...say ready
261D 24 02                      in      al,lstat        ;check for handshake
261F 74 11                      and     al,linsk
2621 E4 04                      jz      LISTST1         ;not yet...say still busy
2623 24 7F                      in      al,ldata        ;got something...
2625 3C 11                      and     al,7fh          ;strip parity
2627 B0 00                      cmp     al,'Q'-40h      ;is it XON?
2629 75 07                      mov     al,0
262B F6 D0                      jnz     LISTST1         ;no, return false
262D C6 06 88 27 00             not     al              ;ready...exit with al=0ffh, zf=0
                                mov     lstactive,0     ;clear list active flag
                        LISTST1:
2632 84 C0                      test    al,al           ;make sure flags are set
2634 C3                         ret

2635 C3                 PUNCH:          ;write punch device
                                ret     ;is a "bit bucket"

2636 B0 1A              READER:
2638 C3                         mov     al,1ah  ;is an EOF source
                                ret

2639 A0 8C 27           GETIOBF:
263C C3                         mov al,IOBYTE
                                ret

263D 88 0E 8C 27        SETIOBF:
2641 C3                         mov IOBYTE,cl   ;set iobyte
                                ret             ;iobyte not implemented

2642 8A 07              pmsg:
2644 84 C0                      mov al,[BX]     ;get next char from message
                                test al,al
```

```
                                   ;*                                          *
                                   ;********************************************
2642 8A 07                                  mov al,[BX]     ;get next char from message
2644 84 C0                                  test al,al
```

```
2646 74 26                          jz return       ;if zero return
2648 8A C8                          mov CL,AL
264A E8 97 FF                       call CONOUT     ;print it
264D 43                             inc BX
264E EB F2                          jmps pmsg       ;next character and loop

                            ;*********************************************
                            ;*                                           *
                            ;*          Disk Input/Output Routines       *
                            ;*                                           *
                            ;*********************************************

                            SELDSK:         ;select disk given by register CL
 0002                       ndisks  equ     2 ;number of disks (up to 16)
2650 C6 06 8D 27 FF                 mov seekfg,0ffh ;set seek flag
2655 88 0E 8E 27                    mov disk,cl     ;save disk number
2659 BB 00 00                       mov bx,0000h    ;ready for error return
265C 80 F9 02                       cmp cl,ndisks   ;n beyond max disks?
265F 73 0D                          jnb return      ;return if so
2661 B5 00                          mov ch,0        ;double(n)
2663 8B D9                          mov bx,cx       ;bx = n
2665 B1 04                          mov cl,4        ;ready for *16
2667 D3 E3                          shl bx,cl       ;n = n * 16
2669 B9 56 28                       mov cx,offset dpbase
266C 03 D9                          add bx,cx       ;dpbase + n * 16
266E C3                     return: ret             ;bx = .dph

                            HOME:   ;move selected disk to home position (Track 0)
266F B9 00 00                       mov cx,0        ;set disk i/o to track zero
                                                    ;**** fall through ****
                            SETTRK: ;set track address given by CX
2672 89 0E 8F 27                    mov trk,CX
2676 C6 06 8D 27 FF                 mov seekfg,0ffh ;set seek flag
267B C3                             ret

                            SETSEC: ;set sector number given by cx
267C 89 0E 91 27                    mov sect,CX
2680 C3                             ret

                            SECTRAN: ;translate sector CX using table at [DX]
2681 8B D9                          mov bx,cx
2683 03 DA                          add bx,dx       ;add sector to tran table address
2685 8A 1F                          mov bl,[bx]     ;get logical sector
2687 C3                             ret

                            SETDMA: ;set DMA offset given by CX
2688 89 0E 93 27                    mov dma adr,CX
268C C3                             ret

                            SETDMAB: ;set DMA segment given by CX
268D 89 0E 95 27                    mov dma seg,CX
2691 C3                             ret
                            ;
                            GETSEGT: ;return address of physical memory table
2692 BB 51 28                       mov bx,offset seg table
2695 C3                             ret

                            ;*********************************************
                            ;*                                           *
                            ;*  All disk I/O parameters are setup:       *
                            ;*      DISK     is disk number     (SELDSK) *
                            ;*      TRK      is track number    (SETTRK) *
                            ;*      SECT     is sector number   (SETSEC) *
                            ;*      DMA ADR  is the DMA offset  (SETDMA) *
                            ;*      DMA SEG  is the DMA segment (SETDMAB)*
                            ;*  READ reads the selected sector to the DMA*
                            ;*  address, and WRITE writes the data from  *
                            ;*  the DMA address to the selected sector   *
                            ;*  (return 00 if successful,  01 if perm err)*
                            ;*                                           *
                            ;*********************************************

                            READ:
2696 B1 0A                          mov cl,10       ;set retry count
                            READ1:
2698 E8 89 00                       call STUP       ;set up unit/track/sector
269B B0 03                          mov al,3        ;send read command
269D E8 D5 00                       call DLOOP
26A0 BA C0 C0                       mov dx,datai    ;set port number
```

```
26A3 EC                             in al,dx        ;get back status
26A4 A8 08                          test al,8       ;check CRC flag
26A6 74 16                          jz RDOK         ;no error...go get data
26A8 FE C9                          dec cl          ;got an error...count retrys
26AA 75 03                          jnz READ2       ;some retrys left...continue
26AC B0 01                          mov al,1        ;bad news....return error
26AE C3                             ret
                            READ2:
26AF F6 C1 03                       test cl,3       ;time for a re-seek?
26B2 7B E4                          jpo READ1       ;no, just reread
26B4 C6 06 8D 27 FF                 mov seekfg,0ffh ;yes, set seek flag
26B9 E8 C6 00                       call RESET      ;clear errors, home drive
26BC EB DA                          jmps READ1      ;try read again
                            RDOK:
26BE B9 80 00                       mov cx,128      ;set byte counter
26C1 FC                             cld             ;set forward direction
26C2 06                             push es         ;save extra segment
26C3 C4 3E 93 27                    les di,dword ptr dma adr        ;set dest index and segment
26C7 BA C0 C0                       mov dx,cntrl
                            RDLUP:
26CA B8 40 00                       mov ax,40h      ;send "examine read buffer" command
26CD EE                             out dx,al       ;to disk control port
26CE EC                             in al,dx        ;get data byte
26CF AA                             stos al         ;store it, bump pointer and count
26D0 B0 41                          mov al,41h      ;send "step read buffer" command
26D2 EE                             out dx,al       ;to controller
26D3 E2 F5                          loop RDLUP      ;repeat 128 times
26D5 07                             pop es          ;restore extra segment
26D6 B0 00                          mov al,0        ;return good status
26D8 EE                             out dx,al       ;also put controller in status mode
26D9 C3                             ret

                            WRITE:
26DA B9 80 00                       mov cx,128      ;set 128 byte counter
26DD FC                             cld             ;set forward direction
26DE 1E                             push ds         ;save current data segment
26DF C5 36 93 27                    lds si,dword ptr dma adr        ;set source index
                            WRLUP:
26E3 AC                             lods al         ;get next byte
26E4 BA C1 C1                       mov dx,datao
26E7 EE                             out dx,al       ;send to controller
26E8 B0 31                          mov al,31h      ;send "shift write buffer" command
26EA BA C0 C0                       mov dx,cntrl
26ED EE                             out dx,al       ;to controller
26EE B0 00                          mov al,0        ;remove command
26F0 EE                             out dx,al       ;(bit 0 must toggle to be seen)
26F1 E2 F0                          loop WRLUP      ;repeat for sector length times
26F3 1F                             pop ds
                            RTRYP:
26F4 E8 2D 00                       call STUP       ;setup for write
26F7 EC                             in al,dx        ;check controller status
26F8 A8 10                          test al,10h     ;write protected?
26FA 74 08                          jz TRYWR        ;no, continue
26FC BB 36 28                       mov bx,offset prtmsg    ;say "protected"
26FF E8 0A 00                       call ERROR      ;and wait for user action
2702 EB F0                          jmps RTRYP      ;retry if user hits return key
                            TRYWR:
2704 B0 05                          mov al,5        ;send write command
2706 E8 6C 00                       call DLOOP      ;to controller with wait
                            WROK:
2709 B0 00                          mov al,0        ;return good status
270B C3                             ret

                            ;*********************************************
                            ;*                                           *
                            ;*          Disk Utility Routines            *
                            ;*                                           *
                            ;*********************************************
                            ;
                            ;print an error message and wait for user response
                            ;if control-c, then abort to cp/m, else return
                            ;to caller and (usually) retry operation
                            ERROR:
270C E8 33 FF                       call PMSG       ;print an error message
270F E8 C8 FE                       call CONIN      ;wait for user response
2712 50                             push ax         ;save character
2713 BB 4E 28                       mov bx,offset crlf      ;echo cr, lf
2716 E8 29 FF                       call PMSG
2719 58                             pop ax          ;now look at char
```

```
271A 3C 03              cmp al,3        ;control-c?
271C 74 01              jz ERR1         ;yes, return to cp/m
271E C3                 ret             ;else retry error'd operation
                ERR1:
271F B1 00              mov cl,0        ;tell cp/m user 0, drive A
2721 E9 DC D8           jmp ccp         ;bye-bye

                ;Perform select and possibly seek logic for either a
                ;read or write operation.
                STUP:
2724 B0 0B              mov al,0bh      ;issue "reset errors" command
2726 E8 4C 00           call DLOOP      ;to controller with wait
2729 A0 8E 27           mov al,disk     ;get drive number
272C B1 06              mov cl,6        ;prepare to shift into
272E D3 E0              shl ax,cl       ;high 2 bits of cmd byte
2730 0B 06 91 27        or ax,sect      ;put sector number in low bits
2734 BA C1 C1           mov dx,datao
2737 EE                 out dx,al       ;send to controller
2738 B0 21              mov al,21h      ;issue "set unit/sector" command
273A E8 38 00           call DLOOP
273D BA C0 C0           mov dx,data1
2740 B9 64 00           mov cx,100      ;set up delay loop
                STUP0:
2743 BB 40 1F           mov bx,8000     ;inner delay loop
                STUP1:
2746 EC                 in al,dx        ;get controller status
2747 A8 20              test al,20h     ;check "drive fail" (ready) flag
2749 74 0D              jz STUP2        ;no problem...continue
274B 4B                 dec bx          ;count down inner delay loop
274C 75 F8              jnz STUP1
274E E2 F3              loop STUP0      ;count down outer delay loop
2750 BB 24 28           mov bx,offset rdymsg    ;timed out...complain
2753 E8 B6 FF           call error      ;and wait for response
2756 EB CC              jmps STUP       ;retry the whole mess
                STUP2:
2758 B0 00              mov al,0        ;clear seek flag
275A 86 06 8D 27        xchg al,seekfg  ;and fetch previous value
275E 84 C0              test al,al      ;was it set?
2760 75 01              jnz stup3       ;yes, go do seek or home
2762 C3                 ret             ;no seek needed...exit
                STUP3:
2763 A1 8F 27           mov ax,trk      ;look at track number
2766 84 C0              test al,al      ;is it 0?
2768 74 18              jz RESET        ;yes, do a home
276A BA C1 C1           mov dx,datao
276D EE                 out dx,al       ;otherwise, set new track
276E B0 11              mov al,11h      ;give "set track" command
2770 E8 02 00           call DLOOP
2773 B0 09              mov al,9        ;then give "seek" command
                                        ;**** fall through ****
                ;
                ;This routine issues a controller command and waits for
                 completion
                DLOOP:
2775 BA C0 C0           mov dx,cntrl
2778 EE                 out dx,al       ;send command
2779 B0 00              mov al,0        ;strobe it off
277B EE                 out dx,al
                LOOP1:
277C EC                 in al,dx        ;get controller status
277D A8 01              test al,1       ;check ready bit
277F 75 FB              jnz LOOP1       ;loop till ready
2781 C3                 ret             ;then exit
                ;
                ;This routine issues a "clear" command followed by a "home"
                 command
                RESET:
2782 B0 81              mov al,81h      ;send "clear"
2784 E8 EE FF           call DLOOP
2787 B0 0D              mov al,0dh      ;send "home"
2789 EB EA              jmps DLOOP


                ;*****************************************
                ;*                                       *
                ;*              Data Areas               *
                ;*                                       *
                ;*****************************************
```

```
278B                    data offset     equu offset $

                                dseg
                                org     data offset     ;contiguous with code segment
278B 00                 lstactive       db      0       ;set if list handshake active
278C 00                 IOBYTE  db      0       ;i/o assignments (unused at present)
278D 00                 seekfg  db      0       ;set to 0ffh if next access requires seek
278E 00                 disk    db      0       ;disk number
278F 00 00              trk     dw      0       ;track number
2791 00 00              sect    dw      0       ;sector number
2793 00 00              dma adr dw      0       ;DMA offset from DS
2795 00 00              dma seg dw      0       ;DMA Base Segment

2797 0D 0A 0D 0A        signon  db      cr,lf,cr,lf
279B 43 50 2F 4D 2D 38          db      'CP/M-86 Version 1.0 for iCom 3712',cr,lf
     36 20 56 65 72 73
     69 6F 6E 20 31 2E
     30 20 66 6F 72 20
     69 43 6F 6D 20 33
     37 31 32 0D 0A
27BE 53 79 73 74 65 6D          db      'System Generated 04/30/81'
     20 47 65 6E 65 72
     61 74 65 64 20 30
     34 2F 33 30 2F 38
     31
27D7 0D 0A 00                   db      cr,lf,0

27DA 0D 0A              int trp db      cr,lf
27DC 49 6E 74 65 72 72          db      'Interrupt Trap Halt at ',0
     75 70 74 20 54 72
     61 70 20 48 61 6C
     74 20 61 74 20 00

27F4 0D 0A              int0 trp db     cr,lf
27F6 44 69 76 69 64 65          db      'Divide Trap Halt at ',0
     20 54 72 61 70 20
     48 61 6C 74 20 61
     74 20 00

280B 0D 0A              int4 trp db     cr,lf
280D 4F 76 65 72 66 6C          db      'Overflow Trap Halt at ',0
     6F 77 20 54 72 61
     70 20 48 61 6C 74
     20 61 74 20 00

2824 0D 0A              rdymsg  db      cr,lf
2826 44 72 69 76 65 20          db      'Drive not ready',0
     6E 6F 74 20 72 65
     61 64 79 00

2836 0D 0A              prtmsg  db      cr,lf
2838 44 72 69 76 65 20          db      'Drive write protected',0
     77 72 69 74 65 20
     70 72 6F 74 65 63
     74 65 64 00

284E 0D 0A 00           crlf    db      cr,lf,0

                        ;       System Memory Segment Table

2851 01                 segtable db 1   ;1 segment
2852 DC 02                      dw tpa seg      ;1st seg starts after BIOS
2854 23 0D                      dw tpa len      ;and extends to 0ffff

                                include singles.lib ;read in disk definitions
                        ;               DISKS 2
                        dpbase  equ     $               ;Base of Disk Parameter Blocks
  2856                  dpe0    dw      xlt0,0000h      ;Translate Table
                                dw      0000h,0000h     ;Scratch Area
                                dw      dirbuf,dpb0     ;Dir Buff, Parm Block
                                dw      csv0,alv0       ;Check, Alloc Vectors
                        dpe1    dw      xlt1,0000h      ;Translate Table
                                dw      0000h,0000h     ;Scratch Area
                                dw      dirbuf,dpb1     ;Dir Buff, Parm Block
                                dw      csv1,alv1       ;Check, Alloc Vectors
                        ;               DISKDEF 0,1,26,6,1024,243,64,64,2
                        ;
                        ;         1944:  128 Byte Record Capacity
                        ;          243:  Kilobyte Drive  Capacity
```

```
                        ;          64:  32 Byte Directory Entries
                        ;          64:  Checked Directory Entries
                        ;         128:  Records / Extent
                        ;           8:  Records / Block
                        ;          26:  Sectors / Track
                        ;           2:  Reserved  Tracks
                        ;           6:  Sector Skew Factor
                        ;
  2876                  dpb0    equ     offset $        ;Disk Parameter Block
2876 1A 00                      dw      26              ;Sectors Per Track
2878 03                         db      3               ;Block Shift
2879 07                         db      7               ;Block Mask
287A 00                         db      0               ;Extnt Mask
287B F2 00                      dw      242             ;Disk Size - 1
287D 3F 00                      dw      63              ;Directory Max
287F C0                         db      192             ;Alloc0
2880 00                         db      0               ;Alloc1
2881 10 00                      dw      16              ;Check Size
2883 02 00                      dw      2               ;Offset
  2885                  xlt0    equ     offset $        ;Translate Table
2885 01 07 0D 13                db      1,7,13,19
2889 19 05 0B 11                db      25,5,11,17
288D 17 03 09 0F                db      23,3,9,15
2891 15 02 08 0E                db      21,2,8,14
2895 14 1A 06 0C                db      20,26,6,12
2899 12 18 04 0A                db      18,24,4,10
289D 10 16                      db      16,22
  001F                  als0    equ     31              ;Allocation Vector Size
  0010                  css0    equ     16              ;Check Vector Size
                                DISKDEF 1,0
                        ;
                        ;       Disk 1 is the same as Disk 0
                        ;
  2876                  dpb1    equ     dpb0            ;Equivalent Parameters
  001F                  als1    equ     als0            ;Same Allocation Vector Size
  0010                  css1    equ     css0            ;Same Checksum Vector Size
  2885                  xlt1    equ     xlt0            ;Same Translate Table
                                ENDEF
                        ;
                        ;       Uninitialized Scratch Memory Follows:
                        ;
  289F                  begdat  equ     offset $        ;Start of Scratch Area
289F                    dirbuf  rs      128             ;Directory Buffer
291F                    alv0    rs      als0            ;Alloc Vector
293E                    csv0    rs      css0            ;Check Vector
294E                    alv1    rs      als1            ;Alloc Vector
296D                    csv1    rs      css1            ;Check Vector
  297D                  enddat  equ     offset $        ;End of Scratch Area
  00DE                  datsiz  equ     offset $-begdat ;Size of Scratch Area
297D 00                         db      0               ;Marks End of Module
                        loc_stk rw  32  ;local stack for initialization
  29BE                  stkbase equ offset $

  29BE                  lastoff equ offset $
  02DC                  tpa_seg equ (lastoff+0400h+15) / 16
  0D23                  tpa_len equ 0ffffh - tpa_seg     ; 64K less 16 byte reset
                                                           vector less cp/m size

29BE 00                         db 0    ;fill last address for GENCMD

                        ;*****************************************
                        ;*                                       *
                        ;*          Dummy Data Section           *
                        ;*                                       *
                        ;*****************************************

  0000                          dseg    0       ;absolute low memory
                                org     0       ;(interrupt vectors)
0000                    int0_offset     rw      1
0002                    int0_segment    rw      1
                        ;       pad to overflow trap vector
0004                            rw      6
0010                    int4_offset     rw      1
0012                    int4_segment    rw      1
                        ;       pad to system call vector
0014                            rw      2*(bdos_int-5)

0380                    bdos_offset     rw      1
0382                    bdos_segment    rw      1
                                END
```

```
                        ;
                        ;                       SD.A86
                        ;                  (revised 05/05/81)
                        ;
                        ;                 SUPER DIRECTORY PROGRAM
                        ;                   by Bruce R. Ratoff
                        ;
                        ;Displays the directory of a CP/M disk, sorted alphabetically,
                        ;with the file size in K, rounded to the nearest CP/M block size.
                        ;
                        ;This latest variation on a common theme will automatically adjust
                        ;itself for any block size and directory length.  If the screen fills,
                        ;the program will pause until a key is struck (see NPL and LPS equates
                        ;below).  Total space used and number of files are printed at end.
                        ;
                        ;Command: SD FILENAME.FILETYPE or just SD
                        ;
                        ;Allows '*' or '?' type specifications.  Drive name may also be
                        ;specified.  Ignores "SYS" files unless SOPT is TRUE and 'S' option
                        ;is given (i.e., SD *.* S will print all files).
                        ;
                        ;05/05/81 Fixed division overflow problem in decimal output routine.
                         (BRR)
                        ;
                        ;05/03/81 First 8086 version. (Bruce R. Ratoff)
                        ;
                        ;Based on 'DIRS' by Keith Petersen, W8SDZ
                        ;
                        ;
0000                    FALSE   EQU     0               ;DEFINE LOGICAL FALSE
FFFF                    TRUE    EQU     NOT FALSE       ;DEFINE LOGICAL TRUE
                        ;
FFFF                    SOPT    EQU     TRUE    ;PUT TRUE TO ALLOW 'DIR *.* S' FORM
FFFF                    WIDE    EQU     true    ;PUT TRUE TO ALLOW 4 NAMES ACROSS
FFFF                    user    equ     true    ;print user numbers for cp/m 2.x also?
                        ;
                        ;
0000                    BASE    EQU     0
0100                    TPA     EQU     100H
                        ;
005C                    FCB     EQU     5CH
                        ;
                                IF      WIDE
0004                    NPL     EQU     4       ;NUMBER OF NAMES PER LINE
                                ENDIF
                        ;
                                IF      NOT WIDE
                        NPL     EQU     3       ;NUMBER OF NAMES PER LINE
                                ENDIF
                        ;
0017                    LPS     EQU     23      ;NUMBER OF LINES PER SCREEN
003A                    DELIM   EQU     ':'     ;FENCE (DELIMITER) CHARACTER
                        ;
                                org     TPA
                        ;
                        START:
3100 FC                         cld
                        ;
                                IF      SOPT
0101 A0 6D 00                   mov     al,byte ptr .FCB+17     ;SAVE S OPTION FLAG
0104 2E A2 4A 05                mov     SOPFLG,al               ;(BLANK OR LETTER S)
                                ENDIF
                        ;
0108 2E C6 06 4B 05 00          mov     USERNO,0        ;DEFAULT TO USER 0
010E 2E C6 06 3D 05 00          mov     LINCNT,0        ;CLEAR COUNT OF LINES ON SCREEN
0114 B1 0C                      mov     cl,12
0116 E8 15 04                   CALL    BDOS    ;CHECK CP/M VERSION
0119 2E 89 1E 4E 05             mov     word ptr VERFLG,bx  ;LO ORD >0 IF 2.X, HI ORD>0 IF MP/M
011E B2 FF                      mov     dl,0FFH
0120 B1 20                      mov     cl,CURUSR ;INTERROGATE USER NUMBER
0122 E8 09 04                   CALL    BDOS
0125 2E A2 4B 05                mov     USERNO,al
                        ;
                                if      not user
                                mov     al,MPMFLG       ;MP/M?
                                test    al,al   ;IF SO, TYPE READING LINE
                                JZ      CHKDRV  ; ELSE SKIP IT
```

```
                        endif
                ;
0129 BA 0F 02           mov     dx,offset USRMSG ;DISPLAY IT
012C B1 09              mov     cl,PRINT
012E E8 FD 03           CALL    BDOS    ;FIRST PART OF MESSAGE
0131 2E A0 4B 05        mov     al,USERNO
0135 3C 0A              cmp     al,10   ;IF USER NO. > 9 PRINT LEADING 1
0137 72 0B              JB      DUX
0139 B0 31              mov     al,'1'
013B E8 67 03           CALL    TYPC
013E 2E A0 4B 05        mov     al,USERNO       ;PRINT LOW DIGIT OF USER NO.
0142 2C 0A              sub     al,10
                ;
0144 04 30      DUX:    add     al,'0'
0146 E8 5C 03           CALL    TYPC
0149 BA 23 02           mov     dx,offset USRMS2 ;PRINT TAIL OF MESSAGE
014C B1 09              mov     cl,PRINT
014E E8 DD 03           CALL    BDOS
0151 2E C6 06 3D 05 01  mov     LINCNT,1        ;WE USED A LINE
                ;
0157 BE 5C 00   CHKDRV: mov     si,offset FCB
015A AC                 lods    al      ;get drive name
015B 84 C0              test    al,al   ;ANY SPECIFIED?
015D 75 0A              JNZ     START2  ;YES SKIP NEXT ROUTINE
015F B1 19              mov     cl,CURDSK
0161 E8 CA 03           CALL    BDOS    ;GET CURRENT DISK NR
0164 FE C0              inc     al      ;MAKE A:=1
0166 A2 5C 00           mov     byte ptr .FCB,al
                ;
0169 04 40      START2: add     al,'A'-1        ;MAKE IT PRINTABLE
016B 2E A2 68 04        mov     DRNAM,al        ;SAVE FOR LATER
016F BF 5D 00           mov     di,offset FCB+1 ;POINT TO NAME
0172 8A 05              mov     al,[di] ;ANY SPECIFIED?
0174 3C 20              cmp     al,' '
0176 75 07              JNZ     GOTFCB
                ;No FCB - make FCB all '?'
0178 B9 0B 00           mov     cx,11   ;FN+FT COUNT
017B B0 3F              mov     al,'?'
                ;
017D F3 AA              rep stos al     ;fill fcb with '?'
                ;
                GOTFCB:
017F C6 06 68 00 3F     mov     byte ptr .FCB+12,'?'    ;FORCE WILD EXTENT
0184 A0 5C 00           mov     al,byte ptr .FCB  ;CHECK FOR EXPLICIT DRIVE
0187 FE C8              dec     al
0189 8A D0              mov     dl,al   ;SELECT SPECIFIED DRIVE
018B 81 0E              mov     cl,SELDSK
018D E8 9E 03           CALL    BDOS
0190 C6 06 5C 00 00     mov     byte ptr .FCB,0
                ;
0195 B1 1F              mov     cl,CURDPB;IT'S 2.X OR MP/M...REQUEST DPB
0197 06                 push    es      ;save current extra segment
0198 CD ED              int     224     ;return bx=offset dpb, es=segment dpb
019A 83 C3 02           add     bx,2
019D 26 8A 07           mov     al,es: [bx]
01A0 2E A2 33 05        mov     BLKSHF,al       ;GET BLOCK SHIFT
01A4 43                 inc     bx      ;BUMP TO BLOCK MASK
01A5 26 8A 07           mov     al,es: [bx]
01A8 2E A2 34 05        mov     BLKMSK,al
01AC 83 C3 02           add     bx,2
01AF 26 8B 07           mov     ax,es: [bx]
01B2 2E A3 35 05        mov     BLKMAX,ax
01B6 83 C3 02           add     bx,2
01B9 26 8B 07           mov     ax,es: [bx]
01BC 2E A3 37 05        mov     DIRMAX,ax       ;SAVE IT
01C0 07                 pop     es              ;restore our extra segment
                ;
01C1 40         SETTBL: inc     ax      ;DIRECTORY SIZE IS DIRMAX+1
01C2 D1 E0              shl     ax,1    ;DOUBLE DIRECTORY SIZE
01C4 05 51 05           add     ax,offset ORDER ;TO GET SIZE OF ORDER TABLE
01C7 2E A3 3E 05        mov     T&LOC,ax ;NAME TABLE BEGINS WHERE ORDER
                                        TABLE ENDS
01CB 2E A3 40 05        mov     NEXTT,ax
01CF 8B 1E 06 00        mov     bx,word ptr .BASE+6     ;MAKE SURE WE HAVE
                                        ROOM TO CONTINUE
01D3 3B C3              cmp     ax,bx
01D5 72 03              jb      SFIRST
01D7 E9 A5 00           JMP     OUTMEM
                ;
                        ;Look up the FCB in the directory
                ;
01DA B1 11      SFIRST: mov     cl,FSRCHF ;GET 'SEARCH FIRST' FNC
01DC BA 5C 00           mov     dx,offset FCB
01DF E8 4C 03           CALL    BDOS    ;READ FIRST
01E2 FE C0              inc     al      ;WERE THERE ANY?
01E4 75 4D              JNZ     SOME    ;GOT SOME
                ;
01E6 BA FF 01   NONE:   mov     dx,offset FNF   ;PREPARE MP/M ERROR MESSAGE
01E9 2E A0 4F 05        mov     al,MPMFLG
01ED 84 C0              test    al,al   ;USE IT IF REALLY MP/M
01EF 74 03              jz      NOFILE
01F1 E9 33 03           JMP     ERXIT1
01F4 E8 2F 03   NOFILE: CALL    ERXIT   ;ELSE USE CP/M ERROR MESSAGE
01F7 4E 4F 20 46 49 4C  DB      'NO FILE$'
     45 24
01FF 46 69 6C 65 20 6E FNF      DB      'File not found.$'
     6F 74 20 66 6F 75
     6E 64 2E 24
                ;
020F 44 69 72 65 63 74 USRMSG   DB      'Directory for user $'
     6F 72 79 20 66 6F
     72 20 75 73 65 72
     20 24
0223 3A 0D 0A 24        USRMS2  DB      ':',13,10,'$'
                ;
                ;Read more directory entries
                ;
0227 B1 12      MORDIR: mov     cl,FSRCHN ;SEARCH NEXT
0229 BA 5C 00           mov     dx,offset FCB
022C E8 FF 02           CALL    BDOS    ;READ DIR ENTRY
022F FE C0              inc     al      ;CHECK FOR END (0FFH)
0231 74 60              JZ      SPRINT  ;NO MORE - SORT & PRINT
                ;
                ;Point to directory entry
                ;
0233 FE C8      SOME:   dec     al      ;UNDO PREV 'INR A'
0235 B1 05              mov     cl,5
0237 D2 E0              shl     al,cl   ;entry no. times 32
0239 B4 00              mov     ah,0
023B 04 80              add     al,80h
023D 8B D8              mov     bx,ax ;POINT TO BUFFER
                                      ;(SKIP TO FN/FT)
                ;
                        IF      SOPT
023F 2E A0 4A 05        mov     al,SOPFLG       ;DID USER REQUEST SYS FILES?
0243 3C 53              cmp     al,'S'
0245 74 06              JZ      SYSFOK
                        ENDIF
                ;
0247 F6 47 0A 80        test    byte ptr 10[bx],80H     ;check bit 7 of SYS byte
024B 75 DA              JNZ     MORDIR  ;SKIP THAT FILE
                ;
024D 2E A0 4B 05 SYSFOK: mov    al,USERNO       ;GET CURRENT USER
0251 3A 07              cmp     al,[bx]
0253 75 D2              JNZ     MORDIR  ;IGNORE IF DIFFERENT
0255 43                 inc     bx
                ;
                ;Move entry to table
                ;
0256 8B F3              mov     si,bx   ;si points to name
0258 2E 8B 3E 40 05     mov     di,NEXTT        ;NEXT TABLE ENTRY TO di
025D B9 0C 00           mov     cx,12   ;ENTRY LENGTH (NAME, TYPC, EXTENT)
                ;
0260 AC         TMOVE:  lods    al      ;GET ENTRY CHAR
0261 24 7F              and     al,7FH  ;REMOVE ATTRIBUTES
0263 AA                 stos    al      ;store in table
0264 E2 FA              loop    TMOVE
0266 8A 44 02           mov     al,2[si]        ;get sector count
0269 88 05              MOV     [di],al ;STORE IN TABLE
026B 47                 inc     di
026C 2E 89 3E 40 05     mov     NEXTT,di        ;SAVE UPDATED TABLE ADDR
0271 2E FF 06 42 05     inc     COUNT
0276 83 C7 0D           add     di,13   ;SIZE OF NEXT ENTRY
0279 2B 3E 06 00        sub     di,word ptr .BASE+6     ;PICK UP TPA END
027D 72 A8              JB      MORDIR  ;IF TPA END>NEXTT THEN LOOP BACK FOR MORE
                ;
027F E8 A4 02   OUTMEM: CALL    ERXIT
0282 4F 75 74 20 6F 66  DB      'Out of memory.',13,10,'$'
```

```
     20 6D 65 6D 6F 72
     79 2E 0D 0A 24
                        ;
                        ;Sort and print
                        ;
0293 2E 8B 0E 42 05     SPRINT: mov     cx,COUNT        ;GET FILE NAME COUNT
0298 85 C9                      test    cx,cx
029A 75 03                      jnz     SPRINT
029C E9 47 FF                   jmp     NONE    ;NONE, EXIT
                        ;Init the order table
029F 2E A1 3E 05        SPRINT: mov     ax,TBLOC        ;GET START OF NAME TABLE
02A3 BF 51 05                   mov     di,offset ORDER ;POINT TO ORDER TABLE
                        ;
02A6 AB                 BLDORD: stos    ax
02A7 05 0D 00                   add     ax,13
02AA E2 FA                      loop    BLDORD
02AC 2E 8B 1E 42 05             mov     bx,COUNT        ;GET COUNT
02B1 2E 89 1E 44 05             mov     SCOUNT,bx       ;SAVE AS # TO SORT
02B6 4B                         dec     bx      ;only 1 entry?
02B7 74 38                      JZ      DONE    ;..YES, SO SKIP SORT
                        ;
02B9 2E C6 06 46 05 00  SORT:   mov     SWITCH,0        ;SHOW NONE SWITCHED
02BF 2E 8B 1E 44 05             mov     bx,SCOUNT       ;GET COUNT
02C4 4B                         dec     bx              ;use 1 less
02C5 2E 89 1E 4C 05             mov     word ptr TEMP,bx        ;SAVE # TO COMPARE
02CA 2E 89 1E 44 05             mov     SCOUNT,bx       ;SAVE HIGHEST ENTRY
02CF 74 20                      JZ      DONE    ;EXIT IF NO MORE
02D1 BB 51 05                   mov     bx,offset ORDER ;POINT TO ORDER TABLE
                        ;
02D4 B9 0C 00           SORTLP: mov     cx,12   ;# BYTES TO COMPARE
02D7 E8 36 02                   CALL    COMPR   ;COMPARE 2 ENTRIES
02DA 76 03                      jbe     NOSWAP
02DC E8 39 02                   CALL    SWAP    ;SWAP IF NOT IN ORDER
02DF 83 C3 02           NOSWAP: add     bx,2    ;bump order table ptr
02E2 2E FF 0E 4C 05             dec     TEMP    ;BUMP COUNT
02E7 75 EB                      JNZ     SORTLP  ;CONTINUE
                        ;One pass of sort done
02E9 2E A0 46 05                mov     al,SWITCH       ;ANY SWAPS DONE?
02ED 84 C0                      test    al,al
02EF 75 C8                      JNZ     SORT
                        ;
                        ;Sort is all done - print entries
                        ;
02F1 BB 51 05           DONE:   mov     bx,offset ORDER
02F4 2E 89 1E 40 05             mov     NEXTT,bx
                        ;
                        ;Print an entry
                        ;
                                IF      NOT WIDE
                                CALL    DRPRNT  ;PRINT DRIVE NAME
                                ENDIF
02F9 B9 04 00                   mov     cx,NPL  ;NR. OF NAMES PER LINE
02FC 2E C7 06 39 05 00          mov     TOTSIZ,0        ; TOTAL K USED
     00
0303 2E C7 06 3B 05 00          mov     TOTFIL,0        ; AND TOTAL FILES
     00
                        ;
030A 2E 8B 1E 42 05     ENTRY:  mov     bx,COUNT        ; CHECK COUNT OF
                                                          REMAINING FILES
030F 4B                         dec     bx      ; skip compare if only 1 left
0310 74 2D                      JZ      OKPRNT
0312 51                         PUSH    cx
                        ;
0313 B1 06                      mov     cl,dconio       ;get console status
0315 B2 FF                      mov     dl,0ffh
0317 E8 14 02                   call    bdos
031A 84 C0                      test    al,al   ;char?
031C 74 03                      jz      nobrk   ;no char, bypass the other stuff
031E E9 0B 02                   jmp     exit    ;abort
                        ;
0321 2E 8B 1E 40 05     NOBRK:  mov     bx,NEXTT
0326 B9 0B 00                   mov     cx,11
0329 E8 E4 01                   CALL    COMPR   ;DOES THIS ENTRY MATCH NEXT ONE?
032C 59                         pop     cx
032D 75 10                      JNE     OKPRNT  ;NO, PRINT IT
032F 83 C3 02                   add     bx,2    ;SKIP, SINCE HIGHEST EXTENT COMES
                                                  LAST IN LIST
0332 2E 89 1E 40 05             mov     NEXTT,bx
0337 2E FF 0E 42 05             dec     COUNT   ;COUNT DOWN
033C E9 CB FF                   JMP     ENTRY   ;GO GET NEXT
                        ;
033F 51                 OKPRNT:
                                push    cx
                        ;
                                IF      NOT WIDE
                                CALL    FENCE   ;PRINT FENCE CHAR AND SPACE
                                ENDIF
                        ;
0340 2E 8B 1E 40 05             mov     bx,NEXTT        ;GET ORDER TABLE POINTER
0345 8B 37                      mov     si,[bx]
0347 83 C3 02                   add     bx,2
034A 2E 89 1E 40 05             mov     NEXTT,bx        ;SAVE UPDATED TABLE POINTER
034F B9 08 00                   mov     cx,8    ;FILE NAME LENGTH
0352 E8 60 01                   CALL    TYPCIT  ;TYPC FILENAME
0355 B0 2E                      mov     al,'.'  ;PERIOD AFTER FN
0357 E8 4B 01                   CALL    TYPC
035A B9 03 00                   mov     cx,3    ;GET THE FILETYPC
035D E8 55 01                   CALL    TYPCIT
0360 8A 14                      mov     dl,[si]
0362 B6 00                      mov     dh,0
0364 46                         inc     si
0365 8A 04                      mov     al,[si] ;GET SECTOR COUNT OF LAST EXTENT
0367 B1 04                      mov     cl,4    ;# OF EXTENTS TIMES 16K
0369 D3 E2                      shl     dx,cl
036B 2E 02 06 34 05             ADD     al,BLKMSK       ;ROUND LAST EXTENT TO BLOCK SIZE
0370 B1 03                      mov     cl,3
0372 D2 E8                      shr     al,cl           ;CONVERT FROM SECTORS TO K
0374 B4 00                      mov     ah,0
0376 03 D0                      add     dx,ax   ;add to total K
0378 2E A0 34 05                mov     al,BLKMSK       ;GET SECTORS/BLK-1
037C B1 03                      mov     cl,3
037E D3 E8                      shr     ax,cl           ;CONVERT TO K/BLK
0380 F7 D0                      not     ax              ;USE TO FINISH ROUNDING
0382 23 D0                      and     dx,ax
0384 2E 01 16 39 05             add     TOTSIZ,dx       ;add to total used
0389 2E FF 06 3B 05             inc     TOTFIL  ;INCREMENT FILE COUNT
038E 8B C2                      mov     ax,dx           ;GET BACK FILE SIZE
0390 E8 1B 00                   CALL    DECPRT  ; AND PRINT IT
0393 B0 6B                      mov     al,'k'  ;FOLLOW WITH K
0395 E8 0D 01                   CALL    TYPC
                        ;
                                IF      NOT WIDE
                                CALL    SPACE
                                ENDIF
                        ;
                        ;See if more entries
                        ;
0398 2E FF 0E 42 05             dec     COUNT   ;COUNT DOWN ENTRIES
039D 59                         pop     cx
039E 74 58                      JZ      PRTOTL  ;IF OUT OF FILES, PRINT TOTALS
03A0 49                         DEC     CX      ;ONE LESS ON THIS LINE
03A1 74 05                      jz      DOCRLF
                        ;
                                IF      WIDE
03A3 E8 F5 00                   CALL    FENCE   ;NO CR-LF NEEDED, DO FENCE
                                ENDIF
                        ;
03A6 EB 03                      jmps    NOCRLF
                        ;
03A8 E8 1D 01           DOCRLF: CALL    CRLF    ;CR-LF NEEDED
03AB E9 5C FF           NOCRLF: JMP     ENTRY
                        ;
                        ;Print HL in decimal with leading zero suppression
                        ;
DECPRT:                         ;CLEAR LEADING ZERO FLAG
03AE 2E C6 06 50 05 00          mov     LZFLG,0
03B4 BB E8 03                   mov     bx,1000 ;PRINT 1000'S DIGIT
03B7 E8 11 00                   CALL    DIGIT
03BA BB 64 00                   mov     bx,100  ;ETC
03BD E8 0B 00                   CALL    DIGIT
03C0 BB 0A 00                   mov     bx,10
03C3 E8 05 00                   CALL    DIGIT
03C6 04 30                      add     al,'0'  ;GET 1'S DIGIT
03C8 E9 DA 00                   JMP     TYPC
                        ;
03CB BA 00 00           DIGIT:  mov     dx,0    ;init hi order dividend
03CE F7 F3                      div     bx      ;divide ax by digit value (dx gets rmdr)
03D0 04 30                      add     al,'0'  ;convert to ASCII digit
```

```
                        ;
03D2 3C 30                      cmp     al,'0'  ;ZERO DIGIT?
03D4 75 16                      JNZ     DIGNZ   ;NO, TYPC IT
03D6 2E A0 50 05                mov     al,LZFLG        ;LEADING ZERO?
03DA 84 C0                      test    al,al
03DC B0 30                      mov     al,'0'
03DE 75 12                      JNZ     DIGPR   ;PRINT DIGIT
03E0 2E A0 47 05                mov     al,SUPSPC       ;GET SPACE SUPPRESSION FLAG
03E4 84 C0                      test    al,al   ;SEE IF PRINTING FILE TOTALS
03E6 74 0D                      jz      DIGNP           ;YES, DON'T GIVE LEADING SPACES
03E8 B0 20                      mov     al,' '
03EA EB 06                      JMPS    DIGPR   ;LEADING ZERO...PRINT SPACE
                        ;
03EC 2E C6 06 50 05 FF DIGNZ:   mov     LZFLG,0ffh ;SET LEADING ZERO FLAG SO NEXT
                                                   ZERO PRINTS
03F2 E8 BD 00           DIGPR:  call    TYPC    ;AND PRINT DIGIT
03F5 8B C2              DIGNP:  mov     ax,dx   ;set up remainder for next digit
03F7 C3                         ret
                        ;
                        ;Show total space and files used
                        ;
03F8 2E C6 06 47 05 00 PRTOTL:  mov     SUPSPC,0        ;SUPPRESS LEADING SPACES
                                                          IN TOTALS
03FE E8 C7 00                   CALL    CRLF    ;NEW LINE (WITH PAUSE IF NECESSARY)
                        ;
                                IF      WIDE
0401 BA 68 04                   mov     dx,offset TOTMS1 ;PRINT FIRST PART OF
                                                           TOTAL MESSAGE
                                ENDIF
                        ;
                                IF      NOT WIDE
                                mov     dx,offset TOTMS1+1 ;PRINT FIRST PART OF
                                                             TOTAL MESSAGE
                                ENDIF
                        ;
0404 B1 09                      mov     cl,PRINT
0406 E8 25 01                   CALL    BDOS
0409 2E A1 39 05                mov     ax,TOTSIZ       ;PRINT TOTAL K USED
040D E8 9E FF                   CALL    DECPRT
0410 BA 75 04                   mov     dx,offset TOTMS2;NEXT PART OF MESSAGE
0413 B1 09                      mov     cl,PRINT
0415 E8 16 01                   CALL    BDOS
0418 2E A1 3B 05                mov     ax,TOTFIL       ;PRINT COUNT OF FILES
041C E8 8F FF                   CALL    DECPRT
041F BA 7B 04                   mov     dx,offset TOTMS3;TAIL OF MESSAGE
0422 B1 09                      mov     cl,PRINT
0424 E8 07 01                   CALL    BDOS
0427 B1 1B                      mov     cl,GALLOC       ;GET ADDRESS OF
                                                          ALLOCATION VECTOR
0429 06                         push    es      ;save our ES
042A CD E0                      int     224     ;return bx=offset ALV, es=segment ALV
042C 2E 8B 16 35 05             mov     dx,BLKMAX       ;GET ITS LENGTH
0431 42                         inc     dx
0432 B9 00 00                   mov     cx,0    ;INIT BLOCK COUNT TO 0
                        ;
0435 53                 GSPBYT: PUSH    bx      ;SAVE ALLOC ADDRESS
0436 26 8A 07                   mov     al,es: [bx]
0439 B3 08                      mov     bl,8    ;SET TO PROCESS 8 BLOCKS
                        ;
043B D0 E0              GSPLUP: shl     al,1            ;TEST BIT
043D 72 01                      JB      NOTFRE
043F 41                         inc     cx
                        ;
0440 4A                 NOTFRE: dec     dx      ;COUNT DOWN BLOCKS
0441 74 08                      JZ      ENDALC  ;QUIT IF OUT OF BLOCKS
0443 FE CB                      dec     bl      ;COUNT DOWN 8 BITS
0445 75 F4                      JNZ     GSPLUP  ;DO ANOTHER BIT
0447 5B                         POP     bx      ;BUMP TO NEXT BYTE
0448 43                         INC     bx      ;OF ALLOC. VECTOR
0449 EB EA                      JMPS    GSPBYT  ;PROCESS IT
                        ;
044B 07                 ENDALC: pop     es      ;restore our es
044C 8B C1                      mov     ax,cx
044E 2E 8A 0E 33 05             mov     cl,BLKSHF       ;GET BLOCK SHIFT FACTOR
0453 80 E9 03                   sub     cl,3    ;CONVERT FROM SECTORS TO K
0456 74 02                      JZ      PRTFRE  ;SKIP SHIFTS IF 1K BLOCKS
                        ;
0458 D3 E0                      shl     ax,cl   ;mult blks by k/blk
                        ;
045A E8 51 FF           PRTFRE: CALL    DECPRT  ;PRINT K FREE
045D BA 88 04                   mov     dx,offset TOTMS4
0460 B1 09                      mov     cl,PRINT
0462 E8 C9 00                   CALL    BDOS
0465 E9 C4 00                   JMP     EXIT    ;ALL DONE...RETURN TO CP/M
                        ;
0468 20 3A 20 54 6F 74  TOTMS1  DB      ' : Total of $'
     61 6C 20 6F 66 20
     24
0468                    DRNAM   equ     TOTMS1
0475 6B 20 69 6E 20 24  TOTMS2  DB      'k in $'
047B 20 66 69 6C 65 73  TOTMS3  DB      ' files with $'
     20 77 69 74 68 20
     24
0488 6B 20 73 70 61 63  TOTMS4  DB      'k space remaining.$'
     65 20 72 65 6D 61
     69 6E 69 6E 67 2E
     24
                        ;
                        FENCE:
                                IF      WIDE
049B E8 05 00                   CALL    SPACE
                                ENDIF
049E B0 3A                      mov     al,DELIM        ;FENCE CHARACTER
04A0 E8 02 00                   CALL    TYPC    ;PRINT IT, FALL INTO SPACE
                        ;
04A3 B0 20              SPACE:  mov     al,' '
                        ;
                        ;Type character in A
                        ;
04A5 51                 TYPC:   PUSH    cx
04A6 52                         PUSH    dx
04A7 53                         push    bx
04A8 56                         push    si
04A9 8A D0                      mov     dl,al   ;use bdos calls, that's what they're there for
04AB B1 06                      mov     cl,dconio
04AD E8 7E 00                   call    bdos
04B0 5E                         pop     si
04B1 5B                         POP     bx
04B2 5A                         POP     dx
04B3 59                         POP     cx
04B4 C3                         RET
                        ;
04B5 AC                 TYPCIT: lods    al
04B6 E8 EC FF                   CALL    TYPC
04B9 E2 FA                      loop    TYPCIT
04BB C3                         RET
                        ;
                        ;Fetch character from console (without echo)
                        ;
04BC B1 06              CINPUT: mov     cl,dconio
04BE B0 FF                      mov     al,0ffh
04C0 E8 6B 00                   call    BDOS
04C3 24 7F                      and     al,7FH
04C5 74 F5                      jz      CINPUT
04C7 C3                         RET
                        ;
04C8 2E A0 3D 05        CRLF:   mov     al,LINCNT       ;CHECK FOR END OF SCREEN
04CC FE C0                      inc     al
04CE 3C 17                      cmp     al,LPS
04D0 72 0D                      JB      NOTEOS  ;SKIP MESSAGE IF MORE LINES LEFT ON SCREEN
04D2 BA F1 04                   mov     dx,offset EOSMSG;SAY WE'RE PAUSING FOR INPUT
04D5 B1 09                      mov     cl,PRINT
04D7 E8 54 00                   CALL    BDOS
04DA E8 DF FF                   CALL    CINPUT  ;WAIT FOR CHAR.
04DD B0 00                      mov     al,0    ;SET UP TO ZERO LINE COUNT
                        ;
04DF 2E A2 3D 05        NOTEOS: mov     LINCNT,al       ;SAVE NEW LINE COUNT
04E3 B0 0D                      mov     al,13   ;print cr
04E5 E8 BD FF                   call    TYPC
04E8 B0 0A                      mov     al,10   ;lf
04EA E8 BB FF                   call    TYPC
                        ;
                                IF      NOT WIDE
                                CALL    DRPRNT  ;DRIVE NAME
                                ENDIF
                        ;
04ED B9 04 00                   mov     cx,NPL  ;RESET NUMBER OF NAMES PER LINE
04F0 C3                         RET
```

```
                        ;
04F1 0D 0A 28 53 74 72  EOSMSG  DB      13,10,'(Strike any key to continue)$'
     69 6B 65 20 61 6E
     79 20 6B 65 79 20
     74 6F 20 63 6F 6E
     74 69 6E 75 65 29
     24
                        ;
                                IF      NOT WIDE
                        DRPRNT: mov     al,DRNAM
                                JMP     TYPC
                                ENDIF
                        ;
                        ;Compare routine for sort
                        ;
0510 8B 37              COMPR:  mov     si,[bx]
0512 8B 7F 02                   mov     di,2[bx]
0515 F3 A6                      repe cmps al,al
0517 C3                         ret
                        ;
                        ;
                        ;Swap entries in the order table
                        ;
0518 2E C6 06 46 05 01  SWAP:   mov     SWITCH,1        ;SHOW A SWAP WAS MADE
051E 8B 17                      mov     dx,[bx]
0520 87 57 02                   xchg    dx,2[bx]
0523 89 17                      mov     [bx],dx
0525 C3                         ret
                        ;
                        ;Error exit
                        ;
0526 5A                 ERXIT:  POP     dx      ;GET MSG
                        ;
0527 B1 09              ERXIT1: mov     cl,PRINT
                        ;
0529 E8 02 00           CALLB:  CALL    BDOS    ;PERFORM REQUESTED FUNCTION
                        ;
                        ;(fall into exit)
                        ;
052C B1 00              ;Exit - all done, restore stack
                        ;
                        EXIT:   mov     cl,0    ;exit is via BDOS call 0
                        ;
052E 06                 BDOS:   push    es      ;preserve es thru bdos call
052F CD E0                      int     224     ;call bdos 8086 style
0531 07                         pop     es
0532 C3                         ret
                        ;
                        ;Temporary storage area
                        ;
0533 00                 BLKSHF  DB      0       ;# SHIFTS TO MULT BY SEC/BLK
0534 00                 BLKMSK  DB      0       ;SEC/BLK - 1
0535 00 00              BLKMAX  DW      0       ;HIGHEST BLOCK # ON DRIVE
0537 00 00              DIRMAX  DW      0       ;HIGHEST FILE # IN DIRECTORY
0539 00 00              TOTSIZ  DW      0       ;TOTAL SIZE OF ALL FILES
053B 00 00              TOTFIL  DW      0       ;TOTAL NUMBER OF FILES
053D 00                 LINCNT  DB      0       ;COUNT OF LINES ON SCREEN
053E 00 00              TBLOC   DW      0       ;POINTER TO START OF NAME TABLE
0540 00 00              NEXTT   DW      0       ;NEXT TABLE ENTRY
0542 00 00              COUNT   DW      0       ;ENTRY COUNT
0544 00 00              SCOUNT  DW      0       ;# TO SORT
0546 00                 SWITCH  DB      0       ;SWAP SWITCH FOR SORT
0547 FF                 SUPSPC  DB      0FFH,   ;LEADING SPACE FLAG FOR DECIMAL RTN.
0548 80 00              BUFAD   DW      BASE+80H ;OUTPUT ADDR
054A 00                 SOPFLG  db      0       ;SET TO 'S' TO ALLOW SYS FILES TO PRINT
054B 00                 USERNO  db      0       ;CONTAINS CURRENT USER NUMBER
054C 00 00              TEMP    dw      0       ;SAVE DIR ENTRY
054E 00                 VERFLG  db      0       ;VERSION FLAG
054F 00                 MPMFLG  db      0       ;MP/M FLAG
0550 00                 LZFLG   db      0       ;0 WHEN PRINTING LEADING ZEROS
  0551                  ORDER   EQU     $       ;ORDER TABLE STARTS HERE
                        ;
                        ;BDOS equates
                        ;
  0001                  RDCHR   EQU     1       ;READ CHAR FROM CONSOLE
  0002                  WRCHR   EQU     2       ;WRITE CHR TO CONSOLE
  0006                  DCONIO  EQU     6       ;direct console i/o
  0009                  PRINT   EQU     9       ;PRINT CONSOLE BUFF
  000B                  CONST   EQU     11      ;CHECK CONS STAT
  000E                  SELDSK  EQU     14      ;SELECT DISK
  000F                  FOPEN   EQU     15      ;0FFH=NOT FOUND
  0010                  FCLOSE  EQU     16      ;   "   "
  0011                  FSRCHF  EQU     17      ;   "   "
  0012                  FSRCHN  EQU     18      ;   "   "
  0019                  CURDSK  EQU     25      ;GET CURRENTLY LOGGED DISK NAME
  001B                  GALLOC  EQU     27      ;GET ADDRESS OF ALLOCATION VECTOR
  001F                  CURDPB  EQU     31      ;GET CURRENT DISK PARAMETERS
  0020                  CURUSR  EQU     32      ;GET CURRENTLY LOGGED USER NUMBER (2.x ONLY)
                        ;
                                END
```

The TEC-86 16-Bit Computer System

Chris Terry

The TEC-86 computer system, manufactured by TecMar Inc., is a general-purpose microcomputer system using the new Intel 8086 16-bit microprocessor. The rugged metal enclosure houses a heavy-duty power supply, an S-100 motherboard with twelve slots, and two Shugart SA800 8" floppy disk drives. The basic system is supplied with:

- CPU board equipped with an Intel 8086 microprocessor running at 5 MHz (4 or 8 MHz options available), an 8259A priority interrupt chip, and power-on jump circuitry;
- 32K of 300nS static RAM on two 16K boards, expandable to 1 Megabyte; available as an option is a single 64K dynamic RAM board at the same price as four 16K boards.
- PROM I/O board equipped wih two 8251A serial ports capable of handling synchronous or asynchronous RS-232 data links at transmission speeds of up to 19,200 baud, an 8255 chip that provides 24 lines of parallel I/O, and sockets for 2K x 16 of PROM;
- Microbyte single/dual density disk controller, based on the NEC 765 LSI controller chip and capable of supporting up to four drives.

The price for the basic system is $3990; additional 16K memory boards are available at $395 each.

Hardware Documentation

The manuals supplied by TecMar for each board in the system are very good. They supply complete logic diagrams which, though reduced to half the original size, are clean and readable, as regards both lettering and layout. They are also split into convenient one page chunks, each of which contains one or more complete functions; connections that have to cross page boundaries are brought to the left or right edge of the diagram and are plainly visible. Pin connections and cabling to the outside world are clear and have text clarifications where necessary. On-board jumpers to select options are similar to those found on disk drives—contact pins which are connected together by jumper connectors in plastic covers. The placement of jumpers is both described and illustrated for each option, and the user should have no difficulty in setting up or changing the jumpers correctly. Switch settings are defined as "Open" or "Closed" according to the marking on the switches, and there are clear statements as to whether a switch closure represents a 1 or a 0 on the associated line.

The theory sections contain enough detail to clue in a person who already has a fair amount of hardware experience, and are enhanced by simplified logic diagrams of functions that might otherwise be difficult to understand. This is a most welcome change from so many other manuals where highly detailed and dense descriptions refer to equally dense fold-outs, with no clue as to where in the drawing to look.

TecMar is to be congratulated on these manuals. They have obviously hired professional writers and given them reasonable time and budget to do a first class job. The language is just informal enough to be readable without losing exactness, and clarity has been made a prime goal.

I found only one typographic error (the notorious "intergrated" chips, which conjures up visions of elves diligently grating cheese into the inter-chip spaces). And only one factual error—which in any case is not calamitous—the I/O board manual calls out RS232 signal levels as +5 to +15 volts for a Mark (1) and -5 to -15 volts for a Space (0). In fact, the RS232-C spec defines the signal level limits as 3 volts to 25 volts in either direction relative to signal ground; the positive level is a SPACE (0) for a data line and ON for a control line, whereas the negative level is a MARK (1) for a data line and OFF for a control line.

The Software

Software to support the TEC-86 consists of CP/M-86 from Digital Research, Inc., and Basic-86 from Microsoft, Inc. TecMar also has Pascal/M-86 from Sorcim available as an option. Mention is made in the PROM I/O board manual of a system monitor for which the PROM sockets are intended, but this does not appear on the current price list. The PROM in the evaluation system contains the CP/M-86 bootstrap and disk primitives, but no monitor accessible to the programmer. It would not be necessary, since the CP/M-86 DDT is perfectly adequate for this purpose.

CP/M-86

This operating system is functionally equivalent to CP/M Version 2.X for the 8080/Z80 systems. The differences are due mainly to the use of separate memory segments for code, data, and stack, and the addition of function calls—CP/M-86 has 59 function codes, compared to the 36 of CP/M-80 Version 2.X. Page 0 is used for the same purposes as in CP/M-80, but the operating system is usually loaded at 400H, directly above the interrupt locations. You can, however, change this location. Relocatable transient programs load above the operating system, starting at 2A00H. Unlike CP/M-80, CP/M-86 does not use absolute locations for system entry or default variables; instead, entry to BDOS takes place through a software interrupt, and entry to BIOS is by a new function call. Most of the new function calls are related to the allocation or releasing of memory.

Because of the additional BDOS functions and a larger BIOS, CP/M-86 is too large to fit on two single-density tracks, though it fits comfortably on two double-density tracks. If single-density is used, the bootstrap loads only the cold-start loader; this in turn loads CP/M-86 from the file area (not the system tracks). A warm start is somewhat simpler than in CP/M-80, since you are not required to reload the CCP and BDOS. Further, relocation of the system is somewhat simpler because relocatable code is used. Thus, there is no MOVCPM utility; the only change is to the cold boot, telling it where to start loading the operating system.

The standard system supplied by TecMar is configured to run in a 64Kbyte memory; however, the distribution disk also contains systems to run in 32K or 96K.

CP/M-86 Documentation

As the Duke of Gloucester remarked when presented with Volume 4 of *The Decline & Fall of the Roman Empire*: "Another damned thick, square, book! Always scribble, scribble, scribble! Eh! Mr. Gibbon?" The TecMar system documentation consists of a six page leaflet describing how to boot up the system (simplicity itself—turn on power, hit RESET, put the disk in the A drive, and close the door!), how to format disks for single or double density, and how to copy the system tracks, for which TecMar has provided utilities to suit the Microbyte controller and formats.

Digital Research has been (necessarily) more lavish. In addition to the *Introduction to CP/M Features and Facilities*, *The CP/M 2.2 User's Guide*, and *The Ed User's Manual*, which are standard for all versions, there is a huge amount of completely new material. *The CP/M-86 Reference Guide* has 138 pages, *The ASM-86 User's Guide* has 75 pages, and *The DDT-86 User's Manual* has 19 pages. *The CP/M-86 Reference Guide* is, like most Digital Research manuals, a tough nut to crack. All the required information is there, but it's not always easy to find. The definitions of BIOS routines and BDOS function calls are easy—they are presented in order, concisely, and reasonably clearly. It's the mass of other information that causes me trouble. I wish I knew why. I cannot complain that the manuals are badly written or disorganized. Individual sentences are perfectly clear, and there is organization. But it always takes me more time than I like to find what I am looking for. What is frustrating is that I cannot think of just how the manual could be better organized. I suppose you just have to read and read and read until you know it almost by heart, and then your brain goes "Click!" and the pieces drop into the places in your brain from which you can most easily retrieve them. Perhaps an index would help?

Performance

For me, the TecMar system has behaved in an exemplary way. I unpacked it, spent three or four hours with the manuals, plugged it in, connected a Lear-Siegler ADM-3A terminal set for 19,200 baud (as instructed), booted up, and away we went. Operationally, the instructions were clear and simple. Except for copying single-density Basic-86 to a double-density working disk, which gave me a little trouble at first, it's just like running CP/M 2.2 and Basic-80.

I have not yet found a huge increase in speed, but that is because I have not yet gotten to any real number-crunching in A86. Basic-80, as I understand, is a simple translation of the interpreter from 8080 language to 8086 language, without optimization to make use of the special features of the 8086 CPU and architecture. Thus, when I loaded my Basic program for testing sorting routines, the interpreter (which runs on a 5-MHz clock) executed Bubble, Heap, Shell-Metzner, and Quick sorts in a shade less than half the time it takes on my 2 MHz 8080 machine using Basic-80. For 200 random numbers, the Bubble sort took 148 seconds instead of 310, Heap took 32 instead of 67, Shell-Metzner took 34 instead of 71, and Quick took 17 instead of 34 (average of three runs each). But I suspect that a Z80 running at 4 MHz would have done nearly as well.

However, I am sure that the speed advantages will be seen when there is more software around that is optimized for the 8086. A nice screen editor like Wordmaster, for example. ED is for the birds unless you still have a Teletype, and I am thankful to hear that impending CP/M-80 Version 3.X will have a screen editor. If an 8086 version also appears that uses the magnificent string handling capability of the 8086, it will probably be a joy to use.

Conclusions

The TEC-86 is rugged, easy to get going, has given me no hardware problems and only minor software puzzlement (I didn't read the manual carefully enough to start with). A price tag of $4600 (which includes 64K of RAM, CP/M-86 and Basic-86) is probably too much for the average hobbyist. But for a small business or a professional user it will be extremely good value, once the software starts being available. And don't forget that there is much more available right now than you might think—you can run any existing Basic-80 program on the 8086, provided that you save it on a single-density disk as ASCII source code. As you may have gathered, I like TecMar's product and their hardware manuals. I wish I could afford it for myself!

Available from: TecMar, Inc., 23600 Mercantile Rd., Cleveland, OH 44122, (216)382-7599.

Chapter VII
Software Directory

Software Directory

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 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: ACCOUNT81
Hardware System: Alpha Micro System
Language: AMOS operating system
Description: Provides accountants with a full inventory of reports and bookkeeping records: chart of accounts, balance sheets, income statements, income journal register, adjustment journal register, G/L, working trial balance, comparison reports, check register, master payroll report, general ledger, employee list, 941's, W-2's, cover letter, check writer and more. Has specially designed input routines that reduce operator fatigue through the use of formatted screens (menus), protected fields and automatic repetition of appropriate data. Smooth, error-free operation is insured by file verification routines that display all incorrectly entered transactions.
Release: February 1981
Price: $1995; updates $295/yr.
Author: Skill Services Inc. of Miami, Fla.
Where to purchase it:
Pony Express Services
100 West 57th St.
New York, NY 10019

Program Name: ACCESS/80 - Information Management System
Hardware System: CP/M Operating System
Minimum Memory Size: 54 K+
Language: Assembly
Description: ACCESS/80 is a high-level, non-programmer oriented system for report generation, data entry, file update, reorganization and maintenance, statistical tabulation, and applications development. Its high level funtionality is comparable to the RAMIS system on IBM mainframes. In addition to functioning as a self-contained system, ACCESS/80 will produce reports from any external file stored in ASCII character format, including Basic and Fortran files.
Price: $795
Included with price: diskette containing program and sample applications; User's Manual, 3 copies of Command Reference Card
Author: Friends Software, Inc.
Where to purchase it:
Friends Software
2020 Milvia Street, Suite 400
P.O. Box 527
Berkeley, CA 94701
(415)540-7282

Program Name: Alpha APL Version 2.0
Hardware System: Alpha Micro
Language: Assembler
Description: Implementation of APL language functionally compatible with large mainframes. Runs as a multi-user system. Has system variables, system functions, I-beam, component I/O and other features. Runs under Alpha Micro operating system. Can be used with either ASCII or APL terminals. Assembler subrontines can be called directly. Source code for many external subroutines and assembler subroutine development aids are included.
Release: November 1980
Price: $500; manual $25
Included with price: Disk and user's manual
Where to purchase it:
Softworks Limited
607 W. Wellington
Chicago, IL 60657

Program Name: Alpha FORTRAN
Hardware System: Alpha Micro (16-bit)
Minimum Memory Size: 32K user memory
Language: Assembler
Description: A multi-user Fortran 77 implementation that has mainframe features. The compiler produces actual assembly language code, not pseudo code, thus allowing Fortran programs to execute many times faster than Basic. Compilations can be stored into a program library, and later linked with assembler or Pascal programs. In addition, Fortran programs are directly callable from Softwork s AlphaAPL language or from Basic. Floating point hardware provides the user with 11 digit accuracy
Releases: April 1981
Price: $600
Included with price: Language, documentation, sample programs
Where to purchase it:
Softworks Limited
607 W. Wellington
Chicago, IL 60657
(312)327-7666

Program Name: APL
Hardware System: 8080/8085/Z80 CP/M
Minimum Memory Size: 44K
Description: Implementation of most of the APL functions and functions of full APL, including n-dimensional inner and outer product, reduction, compression, general transpose, reversal, take, drop; execute and format, system functions and variables, system commands. Runs in either ASC II or bit-pairing ASC II-APL character sets. Can run with user-supplied I/O drivers. Shared variable mechanism allows CP/M disk I/O. Uses Abrams descriptor calculus and shared data storage to save memory space and execution time. Comes with optional driver program for video display with programmable character generator.
Release: October 1980
Price: $350 (NJ residents add 5% sales tax)
Included with price: CP/M disk and Users Manual
Author: Erik T. Mueller
Where to purchase it:
Softronics
36 Homestead Lane
Roosevelt, NJ 08555

Program Name: APL Version 2.3
Hardware System: CP/M system
Minimum Memory Size: 48K
Language: 8080 Machine Code
Description: An APL implementation having most of the functions and operators of full APL, including n-dimensional inner and outer product, reduction, compression, general transpose, reversal, take, drop, execute, format, logarithm, exponential, power, and the circular functions sine, cosine, tangent and arctangent. It has system variables, system functions and shared variables for CP/M disk I/O. The interpreter will run in ASCII using CP/M standard I/O. In addition, it supports typewriter and bit-pairing ASCII-APL character sets and can run

with user-supplied I/O drivers. A driver for a video display with programmable character generator is included. SOFTRONICS APL uses Abrams' descriptor calculus and shared data storage to save memory space and execution time.
Release: Available now
Price: $350
Included with price: CP/M disk, 112 page user manual which includes an APL tutorial
Where to purchase it:
Softronics
36 Homestead Lane
Roosevelt NJ 08555

Program Name: Apparel Management System
Hardware System: CP/M 48K, 2-8" Drives
Language: CBASIC-2
Description: This system is designed to help management make decisions about their stores. Items to reorder that will still make the season, items to be moved from one store to another and items to be marked down are some of the daily tools provided. A detailed inventory report by department shows inventory information (units, dollars in stock, etc.) and monthly sales information. A monthly analysis is done by store/department showing sales, COGS, profit, annual inventory turns, stock to sales ratio and sales compared with budgets. The annual report follows the key mothly analysis figures for a year, again for your comparison abilities. Other major reports include daily sales by department, yearly budgets and physical inventory taking sheets.
Release: Available now
Price: $960.00
Included with price: User documentation, 31 programs warranty
Author: Keystone System, Inc.
Where to purchase it:
Keystone Systems, Inc.
P.O. Box 767
Spokane, WA 99210

Program Name: BASIC-PACK: Statistics Programs
Hardware System: Run Minimal Basic
Minimum Memory Size: 4-12K, depending on program
Language: Basic
Description: Contains 33 statistical programs written in minimal Basic. The programs are listed and documented in the book *BASIC-PACK: Statistics Programs for Small Computers.* Most of the necessary statistical programs are included for small samples. Programs are available for descriptive statistics, confidence intervals, t-test, chi-square, and two-sample tests. The book contains a description a sample run, and a listing of each program.
Price: Book $16.95
Author: Dennie Van Tassel
Where to purchase it:
Prentice-Hall, Inc.
Englewood Cliffs, NJ 07632

Program Name: BDS C Compiler
Hardware System: Anything supporting CP/M
Minimum Memory Size: 32K or more...the more, the better
Language: 8080 Machine code for 8080's and Z80's
Description: Compiles a good subset of UNIX C directly into relocatable load modules; a linker is provided to create the .COM files. Emphasis on speed and simplicity of compilation. The "C" language is aesthetic and concise—very powerful, yet relatively "low level," allowing the programmer to do just about anything. The compiler has been totally engineered to interact and co-exist with CP/M. It is comprised of two main segments, each about 10K, which operate in sequence to do a compilation. Instant support is always available by phone or mail from the author. No elaborate liscensing BS required.
Release: Currently available
Price: $125.00 ($20 for documentation alone).
Included with price: Compiler, Linker, Library Manager, Libraries containing over 75 utility and standard I/O functions, over 150K of sample sources, utility programs, a telecommunications program and more.
Author: Leor Zolman
Where to purchase it:
Lifeboat Associates
2248 Broadway
New York, NY 10024

Program Name: BEEFUP
Hardware System: Dual Drive CP/M with 132 Col. Printer
Minimum Memory Size: 48K
Language: CBasic2
Description: A cow/calf herd-management performance data system, providing two constantly updated reports. Cowprint shows each significant item of data on every calf of every cow currently in the herd (999 max), with calf ratings. Lifetime cow data is at your fingertips, at the office or in the field, in seconds! Calfprint shows cumulative calf data (1000 males, 1000 females per disk) with ratios by sex and year, plus herd sire summaries and ratings.
Release: October 1980
Price: $350; Manual only $20
Included with price: Disk and manual
Where to purchase it:
St. Benedict's Farm
Box 366,
Waelder TX 78959

Program Name: BILLING
Hardware System: CP/M
Minimum Memory Size: 52K bytes
Language: Microsoft Basic
Description: BILLING is an integrated accounts receivable system capable of managing a large volume of accounts. The balance forward method of posting is used and supports four aging periods. It supports multiple billing cycles, optional interest charges, audit reports, batch transaction proof listings with checksums, totals by transaction code and many other features. BILLING requires the ENTRY, EDIT, UDE-SEL, UDE-PRT, MENUU and SORT application utilities.
Release: Available now.
Price: $195; License Agreement Required
Author: The Software Store
Where to purchase it:
The Software Store
706 Chippewa Square
Marquette, MI 49855

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: 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: CBS Version 1.1
Hardware System: CP/M system with 200K bytes of mass storage
Minimum Memory Size: 48K
Language: Assembler
Description: Customized accounting systems, including payables, receivables, inventory control and order entry, are provided through the new Configurable Business System, (CBS Version 1.1) set up without using any programming language. CBS can be used to define an application such as an inventory control system by specifying master files to describe the inventory, customer and vendor files. Transaction files are used to describe specific activities, ie., purchases, sales, etc.

A simple procedure provided by the entry program is used to enter customer, vendor, inventory sales and purchasing information. After data entry is completed, an update program processes the transactions against the master files, updating account balances and inventory data. CBS features a comprehensive report generator for producing invoices purchase orders, re-order reports, special reports, and mailing labels.

The new enhanced CBS Version 1.1 improvements include the capability to produce and read ASCII data files, thus permitting external programs access to file data for specialized processing and/or preparing input data for updating CBS files. Other new features include: Menu Chaining to enable the user to create a "menu of menus", that permits one main entry point to be used for access to all application routines; batched updating enables the user to update a master data base and create new records in master files—including updating of external data files.
Release: September 1980
Price: $395 with $25 for updates; $40 for documentation
Included with price: Disk with documentation
Where to purchase it:
Lifeboat Associates
1651 Third Avenue
New York, NY 10028

Program Name: COMM 4
Hardware System: CP/M and RS-232 Serial Port with modem.
Minimum Memory Size: 16K
Language: 8080 assembler
Description: Provides a comprehensive menu-driven communications package for users of CP/M operating systems linking to time-sharing or other CP/M systems. Terminal mode supports disk log option. Four file transfer modes perform auto disk paging without data loss, CRC-16 error retransmit, FDX no echo wait option, port-port/FDX/HDX modem. Local functions enable disk DIR, read name, delete, log in, plus control character and console echo switch.
Release: Now
Price: $150 source; $75 object
Included with price: Program and documentation.
Author: Hawkeye Grafix
Where to purchase it:
Hawkeye Grafix
23914 Mobile St.
Canoga Park, CA 91307

Program Name: COMMON
Hardware System : CPM 2.2 single density 8"
Language: ASM
Description: COMMON u:filex creates for the current USER a read-only virtual file pointing into filex of USER u. Now all users can have access to all the common utilities without using up the disk in redundant copies.
Release: January 1981
Price: $29.95
Included with price: 8" CP/M disk with ASM, COM and DOC files.
Where to purchase it:
microMethods
Box G
Warrenton, OR 97146

Program Name: COMMX
Hardware System: CP/M
Minimum Memory Size: 16K
Language: 8080 Assembler
Description: Menu driven communication interface program provides links to time-share services, computer bulletin boards, or other CP/M systems. File transfer modes perform automatic disk accessing without data loss when available memory (determined at sign on) is exceeded. XON/XOFF protocol implemented, with full echoplex required from host using full- or half-duplex modem hardware. CRC16 error handling protocol is invoked between COMMX-to-COMMX user links guaranteeing precise data transfers of any data type. While remaining connected, a local mode provides disk directories, rename, delete, log in new disks, console echo control, control character display, and creation of a disk log file for terminal mode session recordings. On screen dialing and mode select are supported for those modem types.
Release: January 1981
Price: $250 Source, $75 Object
Included with price.: User manual, disk $47.50 extra
Author: Hawkeye Grafix
Where to purchase it:
Hawkeye Grafix
23914 Mobile Street
Canoga Park, CA 91307

Program Name: COMPRESS
Hardware System: CP/M
Minimum Memory Size: 16K
Language: 8080 Assembly
Description: COMPRESS is a group of four programs which perform data compression on ASCII data. There are two .COM files which will compress and decompress disk files, and two .REL files which are in subroutine form and act on memory use. Compression varies depending on the contents of the source file, but normally is in the range of 30% - 80%. Works on all 7 bit ASCII data.
Release: July 1981
Price: $50
Included with price: 8" CP/M disk with REL, COM, and DOC files.
Where to purchase it:
New Jersey Software Services
6 Village Circle
Westfield, NJ 07090

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: Comstar
Hardware System: Double or quad density North Star System
Minimum Memory Size: 32K
Language: Machine Language
Description: Full compiler system for North Stas Basic (type 2) programs. Compiled programs run faster and original source is protected. Variable dimensions and disk file numbers must be decimal constants.
Release: December 1980
Price: $400
Included with price: Documentation and disk
Author: Allen Ashley
Where to purchase it:
Allen Ashley
395 Sierra Madre Villa
Pasdena, CA 91107

Program Name: CONST
Hardware System: North Star & Apple
Language: Basic
Description: This program was written to do quantity and sizing take-offs for residential and small commercial structures. To operate the program, the user has only to answer questions concerning room sizes and type of construction.
Release: July 1981
Price: $75; listing only $60.
Included with price: Diskette, On-line Documentation, Support
Where to Purchase It:
Computing Interface
1918 Carnegie Lane #C
Redondo Beach, CA 90278

Program Name: D80
Hardware System: CP/M with 8" drive
Minimum Memory Size: 24K (CP/M System Size)
Language: 8080, 8085, and Z80 compatible machine code
Description: D80 is a flexible and powerful disassembler for 8080, 8085, and Z80 machine code programs. It accepts a machine code program from disk or memory and produces a disk file of the disassembled code using either the Intel or Zilog mnemonics. In the created source file, D80 will produce a map of the object file, symbol table, up to four types of symbolic labels, and uses the ORG, EQU, and ND pseudo-opcodes. The created source code is not held in memory, therefore very large programs can be disassembled. Also, all information about a disassembly can be stored in a disk file and then reloaded at a later time to pick-up where you left off. Also runs in interrupt and MP/M environments.
Release: November 1980
Price: $85.00(disk and manual); $75.00 (disk only); $10.00(manual only)
Included with price: Disk contains D80 with sample disassemblies and original source code of an included utility
Author: Dennis Gallagher
Where to purchase it:
DG Software
P.O. Box 1035
Iowa City, IA 52244

Program Name: DATABS
Hardware System: CP/M 8"
Minimum Memory Size: 40K
Language: 8080 Object Code
Description: DATABS was inspired by CLU developed at MIT. It is a data abstraction language suitable for control and systems programming. The built-in types are boolean, character, single-byte integer, double-byte integer, and string. Data abstractions allow the implementation of user-defined types using a dynamic storage mechanism. Data abstractions are a step beyond structured programming. Programs created using DATABS are easier to design, understand, and modify. DATABS supports UNIX-style command line arguments and I/O redirection with and . A stream abstraction allows terminal and disk input/output. Disk contains the compiler, built-in type and run-time support library, stream abstraction, and command line processor.
Release: March 1981
Price: $49.50; manual only $10
Included with price: 8" disk and manual
Where to purchase it:
Softronics
36 Homestead Lane
Roosevelt, NJ 08555

Program Name: Data Merge
Hardware System: CP/M Based or TRS-80 level I
Minimum Memory Size: 32K
Language: 8080 machine code
Description: Personalized form letters: names, addresses, etc. can be replaced by entries either from a mail list file or from the keyboard; reports and manuals: Table of contents and alphabetized index printed automatically eliminating the tedious task of manually changing page number; contracts and specifications: Standard or frequently used paragraphs or sections can be stored in disk files and inserted by name when a document is printed; and books and articles: Footnotes collected in the text and printed at the page bottom, chapters kept in separate files and chained together when printed.
Release: January 1979
Price: $195
Included with price: 100-page user manual.
Author: M. Posehn
Where to purchase it:
MicroDaSys—Software
Box 36275
Los Angeles, CA 90036

Program Name: D—Directory and Disk Status
Hardware System: any CP/M system
Minimum Memory Size: 16K
Language: 8080 Assembler
Description: This program works with single or double density systems on any selectable disk drive. The directory is presented in 4 columns sorted into alphabetical order (the number of columns is equate selectable in the source program). The first line contains the following disk information: Disk:? Files:? Entries:? (? left) Space used:? K (? K left).
Release: Available now
Price: $40.00 Source $20.00 Object
Included with price: Program and documentation.
Author: Hawkeye Grafix
Where to purchase it:
Hawkeye Grafix
23914 Mobile St.
Canoga Park, CA. 91307

Program Name: DF
Hardware System: CP/M
Minimum Memory Size: 30K CP/M Configuration
Language: Compiled from C (BDS version)
Description: Shows all differences between two versions of a printable file, such as a source program. Re-synchronizes and continues after reporting deletions and insertions. Comparisons are on a line basis, but you can specify the line delimiter and a character to be ignored so that PCL and other text block files can be compared. As compiled, it can handle differences as large as 8K in files of any length.
Release: Available now
Price: $20 (check or money order)
Included with price: CP/M COM and C files on CUTS cassette or a paper listing. Or send a Micropolis Mod II diskette. Can arrange for conversion to standard 8" diskette for $5 extra. Modem?
Where to purchase:
Richard Greenlaw
251 Colony Ct.
Gahanna, Ohio 43230

Program Name: Diagnostics I
Hardware System: CP/M 5" & 8"
Minimum Memory Size: 24K
Language: Supplied as object only
Description: Comprehensive set of CP/M compatible system check-out programs. Finds hardware errors in system, confirms suspicions, or just gives system a clean bill of health. Tests: Memory, Disk, CPU (8080/8085/Z80), CRT, and printer.
Release: now
Price: $50
Included with price: Complete user manual and Diskette.
Author: SuperSoft Associates
Where to purchase it: Direct from us or dealers everywhere.
SuperSoft
Box 1628
Champaign, IL 61820

Program Name: DisAsmb
Hardware System: PolyMorphic Systems 8813 single density
Minimum Memory Size: 32K (40 recommended
Language: PolyMorphic Basic Versions B08C thru C01I
Description: This program is an 8080A Disassembler which disassembles machine language programs back to assembly language. It reads the system library file for system labels and creates other labels as needed. It outputs to a file and produces re-assemblable formatted output with SYSTEM labels.
Release: Available now
Price: $35
Included with price: Support programs and data files. Also includes a Hexidecimal dumper and some reference files.
Where to purchase it:
Ralph E. Kenyon Jr.
145-103 S Budding Ave
Virginia Beach, VA 23452

Program Name: Disc-tionary
Hardware System: CP/M

Minimum Memory Size: 32K
Language: Z80 machine code
Description: High speed text proofreader. A thirty page document can be checked for errors in less than two minutes. Each unrecognized word can be added to the Disc-tionary, rejected, ignored etc. with a single keystroke. Similar words may also be listed. After proofreading, the Disc-tionary leaves files containing the original unmarked text (BAK file), the marked text, and an alphabetized list of misspellings. As distributed, nearly 50,000 words can be recognized, and expansion allows up to four times that many. Many options including automatic suffix removal are available. All functions are performed by a single menu-driven program for ease of use. Two free "bug-fix" updates are included in the price.
Release: September 1981
Price: $79.00
Included with price: User manual, 8" diskette (manual available separately for $15.00).
Where to purchase it:
Stellarsoft Corporation
841 Blanchette Dr.
East Lansing, MI 48823
(517)332-2459

Program Name: DOS/65
Hardware System: Tarbell Disk Controller, 6502 CPU
Minimum Memory Size: 16K
Language: Machine Code
Description: Disk operating system with features similar to CP/M. In addition to basic operating system, distribution disk contains a powerful disk file text editor; a disk based, two-pass assembler; a debugger; a system generation routine and a number of other transient utilities. Routines are also included which show how to modify Pittman Tiny Basic and a RAM based version of Microsoft Basic for DOS/65 including SAVE and LOAD of programs. Available with several transient starting addresses ranging from $200 to $2000 for compatibility with AIM, SYM, KIM, TIM, OSI, PET, and Apple memory allocations.
Release: January 1981
Price: $100-$150 depending on options or special modes. Manual only $30.
Included with price: 8" disk and manual
Where to purchase it:
DOS/65
1363 Nathan Hale Dr.
Phoenixville, PA 19460

Program Name: EDIT
Hardware System: CP/M
Minimum Memory Size: 52K bytes
Language: Microsoft Basic
Description: Used with UDE ENTRY, provides fast and easy editing, insertion, deletion, and searchs for selected records. An optional audit printout of edit changes is provided.
Release: Available now
Price: $95; License Agreement Required
Included with price: Diskette, manual, examples, support.
Author: The Software Store
Where to purchase it:
The Software Store
706 Chippewa Square
Marquette, MI 49855

Program Name: Encode/Decode I & II
Hardware System: CP/M 5" & 8" disks
Minimum Memory Size: 24K CP/M
Language: Supplied as object only
Description: Complete software security system for CP/M. Transforms data stored on disk into coded text which is completely unrecognizable. Encode/decode supports multiple security levels and passwords. A user defined combination (one billion possible) is used to code and decode a file. Encode/decode is available in two versions: Level I provides a level of security for normal use. Level II provides enhanced security for the most demanding needs.
Release: Now
Price: $50/$100
Included with price: User manual and diskette
Author: SuperSoft Associates
Where to purchase it: Direct from us or dealers everywhere
SuperSoft
Box 1628
Champaign, IL 61820

Program Name: Energy Basic
Hardware System: CP/M 2.2 & I.D.S. Modem
Language: Machine Code
Description: Energy Basic is a high level language designed to simplify implementation of energy management systems and similar applications requiring monitoring of time, elapsed time, temperature, kilowatt demand, digital inputs, and control of devices based on such information. It provides the Basic language constructs including FILL, FOR, GOTO, GOSUB, IF, INPUT, LET, LIST, NEXT, OUT, PRINT, RETURN, REM, RUN, STOP, WAIT, ABS, CALL, EXAM, INP, RND AND SIZE. Special commands and functions include MODE, SET, ANSW, ELAP, ORIG, PSWD, TEMP and TIME. For example, X=TEMP(0) sets X to current temperature at sensor 0; T=TIME sets T to current time of day; SET causes current time of day to be set; ANSW places system modem in auto-answer mode; ORIG causes a data communications call to be established to current Originate telephone number; ELAP(A) returns time which has elapsed since A was set equal to TIME; etc. Energy Basic supports a primary system console device, an optional system printer, and an optional originate/answer modem. Energy Basic is available as a development system on 8" or resident on two 2716 type PROMs for dedicated control applications. The application program may also reside in 2716 type PROM. The Development System version of Energy Basic also supports the following commands and functions: BYE, LOAD, NEW, SAVE, and SIZE. LOAD and SAVE retrieve and store Energy Basic source programs to and from disk storage.
Release: January 1981
Price: $195, User's manual only $10
Included with price: Either 8" disk (P/N EB080) or two 2716 EPROMs (P/N EB010) and user's manual.
Where to purchase it:
International Data Systems, Inc.
P.O. Box 17269
Dulles International Airport
Washington, DC 20041

Program Name: Enhanced I/O Drivers
Hardware System: NorthStar MDS or Horizon
Language: 8080 Machine Code
Description: These enhanced I/O drivers for NorthStar DOS (versions 4 & 5), Lifeboat NorthStar CP/M (versions 1.4 & 2.2), and UCSD Pascal (version 1.5) are field tested. NorthStar DOS can now echo console output to printer, suspend console output until another key is pressed, and reassign console device. I/O drivers are available for serial devices, IMSAI's VIOC, Malibu 160 printer, and a modem attached to a serial port with all remote I/O echoed to the local console. CP/M users now have a full implementation of I/O byte, allowing user to reassign console, list, and reader punch to any of four devices such as CRT printing terminal, high speed printer and modem. Includes ability to use NorthStar computer as intelligent terminal which can send or receive disk files. Special support is provided for IMSAI VIOC and Malibu 160. UCSD Pascal (from NorthStar) can detect which device is being used as console and can detect if IMSAI VIOC is present.
Release: Available now
Price: $50 per driver
Included with price: CP/M disk
Where to purchase it:
Aardvark Computer Solutions
9434 Chesapeake Drive #1210
San Diego, CA 92123
(714)292-8338

Program Name: ENTRY
Hardware System: CP/M
Minimum Memory Size: 52K bytes
Language: Microsoft Basic
Description: The UNIVERSAL DATA ENTRY System provides interactive definition of data files complete with CRT format and prompts. Once defined, data entry becomes a 'fill in the blank' operation. A wide range of validity checks reduces error to a minimum.
Release: Currently available
Price: $195; License Agreement Required
Included with price: Diskette, manual, examples, support
Author: The Software Store
Where to purchase it:
The Software Store
706 Chippewa Square
Marquette, MI 49855

Program Name: FORTH by Timin Engineering, Release 2
Hardware System: 8080/8085/Z-80 CP/M or CDOS
Minimum Memory Size: 24K
Description: Enhanced version of FIG FORTH. A FORTH style editor with twenty commands is included, as well as virtual

memory sub-system for disk I/O. The user may also make permanent additions to the resident FORTH vocabulary. A Z-80/8080 assembler is also included, allowing the user to create new FORTH definitions which compile directly into machine code. The IF...ELSE.., BEGIN...UNTIL, and BEGIN... WHILE... control structures may be included in assembler definitions. Other enhancements include an interleaved disk for disk access. A 1024 byte disk block may be read or written in as little as 1/6 second. Eight of these blocks are maintained in RAM for immediate access and automatically swapped with others on the disk as they are needed.
Released: December 1980
Price: $95, IBM compatible 8" SD disk (other formats $110). Manual only: $20, applicable toward purchase on disk. California residents add 6% sales tax.
Included with price: CP/M compatible disk, user's manual and shipping by mail in U.S.
Authors: Dr. Mitchell E. Timin, and FIG
Where to pruchase:
Timin Engineering Co.
9575 Genesee Avenue, Suite E-2
San Diego, CA 92121
(714) 455-9008

Program Name: FORTH
Hardware System: 8080/8085/Z-80 CP/M or CDOS
Minimum Memory Size: 24K
Description: Enhanced version of FIG* FORTH for CP/M or CDOS users. Supplied on CP/M format diskette, ready to run. A FORTH style editor with 20 commands is included, as well as a virtual memory subsystem or disk I/O. These allow the user to easily create new FORTH software which is permanently stored on diskettes, then loaded when needed. The user may also make permanent additions to the resident FORTH vocabulary. A Z-80/8080 assembler is also included, allowing the user to create new FORTH definitions which compile directly into machine code. All Z-80 or 8080 instructions may be used. The IF. . . ELSE . . . UNTIL, and BEGIN . . WHILE . . . control structures may be included in assembler definitions; these will automatically compile into appropriate machine code. Other enhancements include an interleaved disk format that minimizes the time required for disk access. A 1024 byte disk block may be read or written in as little as 1/6 second. Eight of these blocks are maintained in RAM for immediate access and automatically swapped with others on the disk as they are needed. FIG FORTH was originally defined by the FORTH INTEREST GROUP and is very close to the FORTH-79 international standard.
*FORTH Interest Group
Release: October 1980
Price: $95 IBM compatible 8" single density disk (other disk formats $110)
Included with price: CP/M compatible disk, users manual suitable for beginners or experienced users, and shipping
Authors: Mitchell E. Timin, and FIG
Where to purchase it:
Mitchell E. Timin Engineering Company
9575 Genesee Avenue, Suite E2
San Diego, CA 92121

Program Name: HAM Radio DX Package
Hardware System: 8080/Z80 8 inch CP/M
Minimum Memory Size: 24K
Language: Machine
Description: The Package provides operating information for the HAM DXer. This includes directions (compass heading), bearings (degrees), distance (miles, kilometers, hops), and time differential to the DX station. A paginated listing by prefix is produced. The programs are run on an interactive basis, simply by typing the COM file as a CP/M command. No RAM is taken by the data file as the data is called directly from the disk. The data base files can be edited to any length by the user.
Release: Now
Price: $22
Included with price: .COM files, 370 country data file, 50 state data file, Improved directory utility. All on 8 inch disk.
Author: Ronald J. Finger
Where to purchase it:
FICOMP
3017 Talking Rock Drive
Fairfax, VA 22031

Program Name: HayesSys
Hardware System: PolyMorphic Systems 8813 single density
Minimum Memory Size: 8K (uses system RAM)
Language: 8080A Machine Language
Description: HayesSys is a D.C. Hayes MICROMODEM 100 terminal operating system consisting of two programs and a modified version of the operating system which allows operation of the D.C. Hayes MICROMODEM 100 board. The system includes complete software control of all parameters, auto dial, and all other features of the D.C. Hayes board except auto answer. The system includes the ability to download to disk files and to send files from disk, as well as the ability to log the incoming text to a printer. Available for Exec/78 and for Exec/83 (specify which).
Release: September 1980
Price: $85 postpaid
Included with price: Installation instructions & disk.
Where to purchase it:
Ralph E. Kenyon Jr.
145-103 S. Budding Ave
Virginia Beach, VA 23452

Program Name: HDBS: An Extended Hierarchical Data Base Management System
Hardware System: Z-80, 8080,6502
Minimum Memory Size: 17K plus approx. 3K for buffers (Z-80)
20K plus approximately 3K for buffers (8080)
26K plus approximately 3K for buffers (6502)
Language: Written in assembly language; interfaces with BASIC, COBOL, FORTRAN and assembly language.
Description: HDBS is a data base management system similar to the MDBS system, except that the data structures which can be handled by HDBS are limited to hierarchics. For many applications a hierarchical system will suffice. A limited read/write protection is available in HDBS at the data base file level. HDBS is designed for use by hobbyists and applications programmers with relatively straight-forward data representation needs.
Release: Currently available
Price: $250.00 - $375.00 (Manual only: $35.00)
Included with price: 260 page User's Manual, HDBS.DDL Data Definition Language, HDBS.DMS Data Management System and a sample program
Author: Micro Data Base Systems
Where to purchase it:
Micro Data Base Systems
PO Box 248
Lafayette, IN 47902

Program Name: Information Master
Hardware System: CP/M Operating Sytem
Minimum Memory Size: 32K
Language: Object program only
Description: Information Retrieval Program handling a large body of static information requiring flexible access. This is accomplished by creating a compact index to the text files based on key words or phrases designated by the user. Retrieved data files may be created with any CP/M compatible text editor or user program in a free form format. Main program maintains a dictionary of all key words indexed, and rapidly searches the index on Boolean (AND & OR) combinations of key words. The program directs the user to the disk containing the data. Retrieved data can be displayed, printed or written to another file. The system is ideal for handling abstracts from scientific literature, product literature, record and book collections, correspondence, recipes, and applications where data is not frequently modified but a large base is required.
Release: 1979
Price: $37.50 plus $1.50 shipping and handling
Included with price: Instruction manual, 8" or 5 1/2" disk containing program & sample data base.
Author: William B. Brogden, Island Cybernetics
Where to purchase it:
Elliam Associates
24000 Bessemer Street
Woodland Hills, CA 91367

Program Name: INFORMATION MASTER
Hardware System: 8080 or Z-80 with two or more disk drives
Minimum Memory Size: 32K
Language: CONVERS, a language similar to Forth and Stoic (note: no additional language package is required to run)
Description: Information Master is an information retrieval program for CP/M and CP/M compatible disk operating systems. The user creates free format text entries using his familiar text editor, setting off keywords or phrases with special character sequences. The program scans this text, creates a compact index, and builds a dictionary of all keywords encountered. Searches are made using single keywords or combinations of keywords in "and" and "or"

clauses. A search of a data base with 500 entries typically takes about 12-15 seconds. After matches have been found, all or part of the original text is recovered for listing, viewing, or copying to a new disk file. Distributed on 8" single density floppy and some 5" formats, write for available formats.
Release: Now
Price: $37.50
Included with price: Information Master program with demonstration data base and configuration customizing program on disk. User's Manual.
Author: William B. Brogden
Where to purchase it:
Island Cybernetics
P.O. Box 208
Port Aransas, TX 78373

Program Name: Infomedia System (IMS) Ver 1.1
Hardware System: S-100 (Vector MZ), Micropolis Drives, CRT with cursor controls
Minimum Memory Size: 48K
Language: MDOS Micropolis Basic of CP/M-CBasic2
Description: A menu driven data base and file management system, plus report writer. All user created data base formats, data files, and report formats are listed on a system directory. Allows up to 20 user defined data formats and reports per disk. The user can define up to 24 numeric or alphanumeric fields for file records. File functions include: Create, Delete, Duplicate, Add/Modify, and List. Record functions include: Add, Update, Delete, Scan, List, Sort, Compact, and Create, Delete, Add/Modify, List and Print. User selectable column or label format, titles, fields, subtotals, total, and printing of selectable records. Fields may be specified as mathematical calculations. of other fields (+, -, *, &, /) are supported. IMS stores up to 20 different files and up to 999 records per disk with full file management capability.
Release: January 1980
Price: $195
Included with price: Eight programs and users manual send $3 for brochure and sample print-outs, or $195 plus $2 shopping. CA residents add 6% sales tax.
Where to purchase it:
Investment Analysis Systems
P.O. Box 282
Palos Verdes Estates, CA 90274

Program Name: Inventory Control for Manufacturers (ICM)
Hardware System: CP/M Version 2.2, 2-8" disks
Minimum Memory Size: 56K
Language: PL/I-80
Description: ICM is a comprehensive inventory control system for a manufacturing environment. Standard inventory control functions are implemented. They include maintaining and reporting on the status of the inventory stock as well as maintaining records of all transactions made against part numbers in stock. In addition, ICM supports multi-level bills of material (BOM's), the creation of multiple part number transactions for jobs based on those BOM's, job-tracking, and generation of materials requirements reports.
Release: Available now
Price: $995
Included with price: Object code and user's manual
Author: Microcomputer Consultants
Where to purchase it:
Microcomputer Consultants
P.O. Box T
Davis, CA 95617

Program Name: ISSCAI SYSTEM
Hardware System: Standard CP/M or MP/M
Minimum Memory Size: 8K
Language: 8080 Assembler
Description: This is a set of three programs CAIGEN, TUTOR, and ENROLL which provide, with the use of a system editor, a complete COMPUTER AIDED INSTRUCTION system. CAIGEN formats a editor written text file to the requirements of TUTOR and creates an enrollment file for the course if needed. TUTOR is the heart of the system providing foward and reverse linking of text, prompting for answers even where there might be several that are correct, responding on correct or incorrect answers with replys if wanted, chaining to next lesson, scoring, passwords, comments and several other functions. ENROLL provides complete enrollment file maintenance and teacher monitoring of student progress in a course, lesson by lesson, this program has password level access.
Release: Currently available
Price: $250.00
Included in price: Object of three programs and users manual. System is available for RESALE LICENSE.
Author: G.B. Shaffstall
Where to purchase it:
International Software Service
13050 W. Cedar Drive #15
Lakewood, Colorado 80228

Program Name: Layout
Hardware System: Sol/Helios II, Disks (1)
Minimum Memory Size: 16K
Language: Extended Disk BASIC
Description: Layout saves programming time and effort by formatting, printing, and screen-printing a series of Data File Layouts. File produces uniform header for program description, and an indefinite number of descriptions of variables used in the data file. Provides space for programmer's comments. Excellent programming and reference tool.
Release: Now
Price: $20, includes source on disk and documentation.
Author: J. Brockway
Where to purchase it:
Jerry Brockway
Suite 306, 2909 Bay to Bay
Tampa, FL 33609.

Program Name: Master Disk Catalog
Hardware System: Micropolis 5 1/4" Drives or Single Density 8" CP/M
Minimum Memory Size: 32K
Language: Assembly—8080
Description: The program maintains a record of the files from your disks on a single catalog disk. As you work on programs or files, comments and dates can be put into them which may then be recorded on the catalog disk. This information may be searched for all occurences of particular file names, for text in your comments, or for dates. File names and disk names may include CP/M "wildcard" characters.
Release: May 1981
Price: $35 & postage
Included with price: Disk, Manual
Where to purchase it:
Mendocino Software
P.O. Box 1564
Willits, CA 95490
(707)459-9130

Program Name: Master Ledger
Hardware system: CP/M, 48K, 2-8" Drives
Language: CBASIC-2
Description: Master ledger analyzes your business. It includes 12 month's budgets plus 12 month account history, making possible for any tape of financial comparisons will show management if they are meeting their financial goals. Quarterly and yera-to-date are also available for any period. Ten different journals, general ledger, trial balance, and budgets are some of the other major reports. The input routines were designed for the operator, easy, fast and verifies all input. Special features include a forced audit trail and a forced balancing system.
Release: Available now
Price: $800.00
Included with price: User documentation, 31 computer programs, warranty
Author: Keystone systems, Inc.
Where to purchase it:
Keystone Systems, Inc.
P.O. Box 767
Spokane, WA 99210.

Program Name: MCALL (Micro proto'CALL' & communications program)
Hardware System: 8080 or Z80 under CP/M serial port connected to an acoustic coupler, video display terminal.
Minimum Memory Size: 16K
Description: MCALL provides the following major functions:

1. Time Sharing Terminal emulation.
2. Disk file transfer between PC (Personal Computer) and TSC (Time Sharing Computer) in either direction.
3. Disk file transfer between two PC's with error detection and retransmission.

To perform function 3, no coordination between operators is required. The file to be transferred is specified by the transmitting operator, then the transmit command is issued (ESC T). The specified file is automatically opened at the TX end and created at the RX end. Subsequently, each

file is closed and a message notifies each operator that the transfer was a success (or not) and displays the total retransmission count. The INFOWORLD Software Report Card for MCALL rated: "Ease of Use" and "Support" as excellent; and "Functionality", "Documentation", and "Error Handling" as good.
Release: Currently available
Price: $50.00
Included with price: Operating Instructions Manual (20 pgs) and 8" single density disk with 84K source file and 8K com file.
Author Tim Pugh
Where to purchase it.
Micro-Call Services
9655-M Homestead Ct.
Laurel, MD 20810

Program Name: MDBS: A Full Network Data Base Management System
Hardware System: Z-80, 8080, 6502
Minimum Memory Size: 17K plus approximately 3K for buffers. (Z-80)
20K plus approximately 3K for buffers. (8080)
26K plus approximately 3K for buffers. (6502)
Language: Written in assembly language; interfaces with BASIC, COBOL, FORTRAN and assembly language.
Description: MDBS is a full network data base system expressly designed for microcomputer use. Details of physically storing, sorting, updating and retrieving data are handled by the MDBS system, freeing the programmer from the tedium and complexity of data management tasks. The amount of data stored is limited only by the amount of on-line disk storage available. Up to 254 different types of data records may be processed, each of which can contain up to 255 data fields. Read/Write access protection is provided at the record, field and set levels. Use of the MDBS system can significantly reduce the cost of developing and maintaining data oriented applications programs.
Release: Currently available
Price: $750.00 - $825.00 (Manual only: $35.00)
Included with price: 260 page User's Manual, MDBS.DDL Data Definition Language, MDBS.DMS Data Management System and a sample program
Author: Micro Data Base Systems
Where to purchase it:
Micro Data Base Systems
PO Box 248
Lafayette, IN 47902

Program Name: MDBS.DRS: A Dynamic Restructuring System for MDBS Data Bases
Hardware System: Z-80, 8080,6502
Minimum Memory Size: 19K plus approximately 3K for buffers (Z-80)
23K plus approximately 3K for buffers (8080)
29K plus approximately 3K for buffers (6502)
Language: Written is assembly language; interfaces with BASIC, COBOL, FORTRAN and assembly language.
Description: MDBS.DRS is a system which can be used to alter the structure of an existing MDBS data base. Its primary use is to permit an MDBS user to include new data fields in existing data records, to define new data records or set relationships in the data base or to delete existing fields, records or sets from a data base. These functions can all be performed without the need to dump the data base contents and reload it, saving much time for the data base user.
Release: Currently available
Price: $100.00 (Manual only: $5.00)
Included with price: MDBS.DRS system and manual with sample application program
Author: Micro Data Base Systems
Where to purchase it:
Micro Data Base Systems
PO Box 248
Lafayette, IN 47902

Program Name: MENU
Hardware System: CP/M
Minimum Memory Size: 48K bytes
Language: Microsoft Basic
Description: MENU Job Stream Control links programs together to form a continuous processing sequence. Displays user defined job stream descriptions and help screens. User programs can be incorporated onto a job stream along with Application Utilities to form complete 'turnkey' systems.
Release: Available now.
Price: $95; License Agreement Required
Included with price: Diskette, manual, examples, support
Where to purchase it:
The Software Store
706 Chippewa Square
Marquette, MI 49855

Program Name: Micro Link
Hardware System: 8080/Z80 System & Modem
Minimum Memory Size: 16K
Description: The Micro Link program enables microcomputer users to communicate over telephone lines. Files transmitted automatically. Readable word-wrapped display fitted to any screen width, a host of options with convenient default settings, and simple, fast user commands are Micro Link features. Micro Link supports originate and answer mode, full- and half-duplex and operates at equipment baud rate. Files may be transmitted in character, line or memory block protocol. The program may be used with others in Basic, assembly or other languages.
Release: June 1981
Price: $89
Included with price: Object code and manual. Supplied on 16 sector, 77 track, 5 1/4" disk (Micropolis); 8" CP/M disk.
Where to puchase it:
Wordcraft
c/o Microcomputer Software Associates
1122 B St.
Hayward, CA 94541
(415)534-2212

Program Name: Micro Link
Hardware System: CP/M 1.4 or Micropolis with serial port & modem
Minimum Memory Size: 16K
Description: Micro Link program enables microcomputer users to communicate with each other, large computers and terminals over telephone lines. Files may be prepared in advance and transmitted automatically. The entire two-way record of communication may be recorded in memory and on disk. Features include readable word-wrapped display fitted to any screen width, a host of options with convenient default settings, and simple, fast user commands. Micro Link scans The Source, other data bases and bulletin boards quickly, recording segments that interest the user for review off line. The Micro Link hosts another computer or a terminal. Micro Link supports originate and answer mode, full- and half-duplex and operates at equipment baud rate. Files may be transmitted in character, line or memory block protocol. The program may be used with others in Basic, assembly or other languages.
Release: April 1981
Price: $89
Included with price: 8" or 5 1/4" disk and manual
Where to purchase it:
Wordcraft
c/o Microcomputer Software Associates
1122 B St.
Hayward, CA 94541
(415)534-2212

Program Name: Milestone
Hardware System: CP/M-86 & 80 x 24 display.
Minimum Memory Size: 56K
Description: Project management software package based on critical path network analysis techniques. Useful for any project that can be broken into a series of distinct tasks, each with a duration, a level of manpower and a cost. Automatically lays out each job against a time scale showing which tasks are critical and which can be delayed. Also displays manpower and expenses versus time, as well as totals and project completion data. Original plan can even be altered during the course of a project to reveal impact of any scheduling changes.
Release: June 1981
Price: $295
Included with price: Disk and Manual
Where to purchase it:
Organic Software
1492 Windsor Way
Livermore, CA 94550
(415)455-4034

Program Name: muLISP-79
Hardware System: Standard CP/M
Minimum Memory Size: 20K
Language: LISP language interpreter
Description: Five man-years in the making and extensively tested, the muLISP-79 interpreter makes a truly sophisticated LISP system available to S-100, CP/M users. It is capable of supporting serious AI efforts in such diverse fields as robotics, game playing, language translation, computer algebra, and theorem proving. Fully integrated into CP/M, it features infinite precision arithmetic, flexible program control constructs, an efficient garbage collector, & informative error messages. Most important for serious applications, it uses the most modern techniques to achieve extremely fast execution speeds. Please write The Soft Warehouse for details. We require a License Agreement be signed prior to shipment.

Release: Now
Price: $190
Included with price: On diskette: muLISP-79 .COM file, Utility library file, Trace facility file, Pretty printer file, & a demo game program. Printed: 80 page Reference Manual, fully indexed.
Author: Albert D. Rich
Where to purchase it:
The Soft Warehouse
P.O. Box 11174
Honolulu, HI 96828

Program Name: MULTI-USER CP/M
Hardware System: CP/M with 8" floppy disk
Minimum Memory Size: 48K bank switchable memory
Language: 8080 machine code
Description: Allows up to four terminals to be supported from one 8080/Z-80 computer. Will also support up to four printers. Lock-out is provided for printer use. System requires one copy of CP/M 1.4 and an interrupt board to generate restart 6 every 16-20 MS.. DMA type disk controller is recommended for faster system performance.
Release: Available now
Price: $125.00 List
Included with price: Diskette and Manual
Author: Mark Winkler
Where to purchase it:
Provar, Inc.
6217 Kennedy Avenue
Hammond, Indiana 46323

Program Name: MWP 2.0-MINI WORD PROCESSING
Hardware System: CP/M
Minimum Memory Size: 48K bytes
Language: Microsoft BASIC
Description: Mini Word Processing enables the user to prepare letters, text, mailing labels and envelopes. Information stored in user defined name and address files can be inserted throughout a letter or text. Documents can be assembled from any number of files stored on the disk, and can be printed or displayed on the CRT with user selectable margins, page size, headers, page numbers and insertions.
Release: Available now
Price: $195 Licence Agreement Required
Included with price: Disk, Manual, examples, support
Author: The Software Store
Where to purchase it:
THE SOFTWARE STORE
706 Chippewa Square
Marquette, MI 49855

Program Name: MWP-SEL
Hardware System: CP/M
Minimum Memory Size: 48K bytes
Language: Microsoft Basic
Description: Allows sophisticated selected records. Example: Select all records with 'AMOUNT DUE' greater than or equal to (=) 'CREDIT LIMIT'. You may combine selection criteria with 'AND' and 'OR' for complex task.

Release: Available now
Price: $95; Licence Agreement Required
Included with price: Diskette, manual, examples, support
Author: The Software Store
Where to purchase it:
The Software Store
706 Chippewa Square
Marquette, MI. 49855

Program Name : Order-Right
Hardware System: CP/M Based—CBasic
Minimum Memory Size: 48K
Language.: CBasic-2
Description: Order-Right is an easy to use order entry system that can interact with Inventory, Accounts Receivable and General Ledger. It lets you enter orders using order codes; automatically calculates discounts for different customer types; allows you to specify special discounts; prints a complete invoice including tax and shipping information; prepares confirmation forms and feeds the Data Merge module for special reports and acknowledgements; prints shipping labels, charge card slips, acknowledgements; allows you to enter customer data from Accounts Receivable; and can interact with Inventory and Accounts Receivable.
Release: August 1980
Price: $800
Included with price: User manual and CBasic source code with sample multiforms (invoice/statement/confirmation)
Author: J. Stuppy
Where to purchase it:
MicroDaSys—Software
Box 36275
Los Angeles, CA 90036

Program Name: Plotter Graphics Package
Hardware System: 8080/Z80 CP/M with either Houston Instruments Hiplot or Tektronix 40xx series terminal
Minimum Memory Size: Depends on how many routines are used
Language: Microsoft FORTRAN-80 and MACRO-80
Description: Set of FORTRAN callable subroutines which implement the standard CALCOMP plot routines: PLOTS, PLOT, FACTOR, WHERE, SCALE, LINE, SYMBOL, NUMBER and AXIS. Also includes several additional routines to support log and semi-log plots, with optional grids. All plotting is done through one simple "driver" routine which may be developed for any particular plotter. Drivers currently exist for Houston Instrument Hiplot and Tektronix 40xx series terminal (or equivalent). Entire ASCII character set is supported by SYMBOL routine. Provided as a 'User Library' from which externals may be satisfied at link time. Source code for both drivers are included. Several demonstration programs are included on the disk.
Release: Currently available
Price: Library and source for drivers; $100.00 Source for entire package: $1000.00
Included with price: 8" 3740 CP/M style diskette containing REL files for each routine, source for drivers, pre-built library for Hiplot (via SIO2 port), and several sample programs using package. Enclosed manual describes calling arguments and operation of each routine. Coupon good for $100 off list price of Hiplot, from the Byte Shop of Columbia, S.C.
Author: Lawrence E. Hughes
Where to purchase it:
Mycroft Labs
P.o. Box 6045
Tallahassee, FL 32301

Program Name: PRGM/MAP
Hardware System: CP/M
Minimum Memory Size: 40K bytes
Language: Machine code
Description: Program Map is a cross-reference tool for Microsoft Basic programs. The system produces alphabetical lists of variables, commands functions, constants, quoted strings and line numbers. Use for documentation, conversion, or I/O modifications.
Release: June 1978
Price: $95; License Agreement Required
Included with price: Diskette, manual, examples, support
Author: The Software Store
Where to purchase it:
The Software Store
706 Chippewa Square
Marquette, MI 49855

Program Name: Programmer's Apprentice
Hardware System: 8080/Z80, CP/M
Minimum Memory Size: 56K of RAM
Language: MBasic
Description: A program development that uses a macro-like language to define standard routines used by its code generator to create MBasic source code programs. Creates fully debugged programs in MBasic to provide screen prompted data input, data base management, file maintenance, and report generation. It is a visually oriented system designed to increase productivity and ease development of interactive application and report programs by performing most of the routine drudgery of programming.

To generate a program, first one creates the screen or report template on the CRT via the built-in screen editor. Then the definition modules use the screen/report template to define the various fields' attributes, eliminating most input errors, and selecting which fields will be the record control keys. Finally, the actual MBasic source code is generated.
Release: October 1981
Included with price: Manual, diskette, software including 80 column report generator.
Where to purchase it:
The Software Group
10471 S. Brookhurst
Anaheim, CA 92804
(714)535-5274

Program Name : Promer
Hardware System: Memory map PROM burning card such as Cromemco bytesaver
Minimum Memory Size: 16K CP/M System
Language: 8080 Assembler
Description: Allows user to build image to be burned in a memory buffer, without having to compute relocation factors, load files into alternate areas, transfer images, or append code sections. It will handle most standard size PROM (256, 512, 1024, 2048, and 4096 bytes per PROM), and verifies erasure at startup time. There may be any number of PROMS in use, and at whatever address the user specifies. Partially burned PROMS may be used, and over burns are possible (the user is notified before this occurs). The program builds the burn image in a stand alone buffer, and does all offsetting and relocating automatically. Data may be loaded from existing ROM, areas in RAM, or disk files (both direct image and hex files are supported), as well as keying directly into or patching the image in the burn buffer. Burning and verifying is automatic, with the user selecting the number of burn passes to execute. The entire program is menu driven, with extensive prompting and explanations on the screen during execution.
Release: Currently available
Price: $60 Source, $30 Object
Included with price: Self prompting software, Disk $7.50 extra
Author: Hawkeye Grafix
Where to purchase it:
Hawkeye Grafix
23914 Mobile Street
Canoga Park, CA 91307

Program Name: Property Analysis System (PAS) Ver 2.11
Hardware System: S-100 (Vector MZ), Micropolis Drives, CRT with Cursor controls
Minimum Memory Size: 48K
Language: MDOS-Micropolis Basic or CP/M-CBasic2
Description: Property Analysis System, for both residential and income property. Analyzes the effects of financing, income, expenses, depreciation, taxes, and inflation on the return on investment for nine years. PAS produces a three-page report consisting of initial conditions and a nine year projection of property value, liabilities, equity, gross income, expenses, net income and cash and percentage return on investment before and after taxes. PAS also provides percentage return on equity for before and after taxes. PAS was designed for ease of use without prior computer experience—field tested since mid 1979. All data can be changed at any time with the effects immediately displayed for review, allowing the user to model an investment property and ask "What if?" PAS stores up to 20 different properties per disk with full file management capability (create, delete, read, & write/update).
Price: $195
Included With price: Five programs and 60 pg. users manual, send $3 for brochure and sample printouts, or $195 plus $2 shipping. CA residents add 6% sales tax.
Where to purchase it:
Investment Analysis Systems
P.O. Box 282
Palos Verdes Estates, CA 90274

Program Name: PRO-TYPE Word Processor
Hardware System: CP/M North Star, MECA ALPHA TAPE
Minimum Memory Size 16K
Language: Basex
Description: IMI's PRO-TYPE is a powerful word processor that is easy to learn and simple to use. Its comprehensive 72-page manual will guide you from beginner, to intermediate and on to advanced applications. PRO-TYPE packs all of these convenient features into a single 8K program that supports fully interactive text entry, editing, and print formatting. Works with ANY type of terminal (memory mapped or not). Floating tabs and underlining. Change left and right margin, line spacing while printing, double text buffers for form letters, etc. Multiple print modes (justification, line fill, verify) Embedded "STOP" codes allow special text insertion command mallow special text insertion command macros for repeated command execution.
Release: Available now
Price North Star 5" SD & DD disk (with manual) $25 MECA ALPHA tape (with manual) $75. CP/M 8" disk (with manual) $75 Add $.75 Special 4th Class or $1.50 Special Handling or UPS
Included with price.72 page manual and disk or tape
Author: Paul K. Warme
Where to purchase it.
Interactive Microware
P O Box 771
State College, PA 16801

Program Name: RF
Hardware System: North Star MDS or Horizon
Minimum Memory Size: 8K Bytes
Language: Assembler, distributed as object code
Description: A file renaming utility designed in the spirit of the disk utilities now supplied with North Star. With RF a simple one-line command is sufficient to change a file name; e.g. the command "GO RF ABC, 2 XYZ" changes file ABC on drive 2 to XYZ. RF prompts for missing parameters and generates meaningful error messages, i.e. "NEW-NAME IN USE." File renaming is a high activity function and should not be left to clumsy, error-prone methods.
Release: Currently available
Price: $7.95 ($1.00 documentation only, credited to purchase); check or money order
Included with price: Disk, documentation, postage & handling; specify single or double density
Author: Jim Hendrix
Where to purchase it:
Jim Hendrix
Rt. 1, Box 74-B-1
Oxford, MS 38655

Program Name: RUNIC 1.0 Language Interpreter
Hardware System: 8" CP/M, TRS-80 Model II, H89 or Apple/CP/M
Minimum 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; specify 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: SCREENMASTER
Hardware System: CP/M compatible. Any dumb terminal.
Minimum Memory Size: 48K recommended.
Language: CBasic-2.
Description: A complete screen handler facility, provided in source code for inclusion in programs requiring full/multi screen input. GOSUB SCREENMASTER. Upon return, multiple screens of validated input are available, in memory, for further use by the programmer. Programmer may insert CBasic-2 code at any of the many exits to affect any degree of editing or control, overriding SCREENMASTER editing if desired. The programmer and, optionally, the terminal user have commands: GOTO (field) n, BACK/FORWARD n (fields), NEXT/PRIOR (screen), PRINT (screen), as well as SUBMIT and ABORT. Utilities are provided to create and test screens as well as to configure any dumb terminal.
Release: August 1981
Price: $195 Manual alone $25 Demo disk $10 additional.
Included with price: 90 page user manual, floppy disk with source code, demos and utilities.
Where to purchase it:
Marketing Essentials, Inc.
206 Mosher Ave.
Woodmere, NY 11598
(212) 560-3589

Program Name: SCREENMASTER
Hardware System: 48K CP/M system. Dumbterminal=Hazeltine,ADM3A,TRS-80 II. Others easily accomodated.
Minimum Memory Size: 40K, 48K recommended.
Language: CBasic-2. Distributed in source code.
Description: Intended for programmers only, Screenmaster allows user to describe multi-screen input via data to the program. Program returns an array of responses, edited for validity. Programmer has pre-/post-input and submit exits where editing and control code may be inserted, commands = 90 to m, back n, forward n, prior (screen 0, next (screen), submit etc. End-user can also be given the commands. Flexible design allows any input scheme to b implemented in minutes rather than days.
Release: Available now
Price: $295. Compiled Demo $50. User manual alone $25.
Author: Dr. Laird Whitehill & Joel Wittenburg.
Where to purchase it:
Micro-computer Business Systems
161 W. 75 St.
New York, NY 10023

Program Name: Small-C Compiler
Hardware System: 8080/Z80
Minimum Memory Size: 48K
Language: C
Description: Small-C is a version of the popular high level language C adapted to the CP/M operating system. The compiler (written in C) produces assembly language for ASM or MAC as its output. The compiler supports a subset of C and also allows assembly language to be included within the C source code with its "#asm...#endasm" feature.
Release: Available now
Price: $15 plus shipping
Included with price: Manual, 8" single density CP/M floppy with executable Small-C, full source code for compiler, the runtime library, and a demonstration program Inc.
Author: Ron Cain, adapted for CP/M by The Code Works
Where to purchase it:
The Code Works
Box 550
Goleta, CA 93017

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

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: SORT 2.0
Hardware System: CP/M
Minimum Memory Size: 48K bytes
Language: Microsoft Basic
Description: General purpose disk sort/merge system for sequential files. User defined SORT task can sort on any number of fields, located anywhere on the record, on ascending or descending sequence.
Release: Currently available
Price: $295; License Agreement Required
Included with price: Diskette, manual, examples support
Author: The Software Store
Where to purchase it:
The Software Store
706 Chippewa Square
Marquette, MI 49855

Program Name: SPDES
Hardware System: North Star
Minimum Memory Size: 16K
Language: Basic
Description: The design of a small signal RF amplifier using S-parameters. Calculation of load and source reflection coefficients; gain and stability circle calculations and analysis; single frequency matching network design.
Release: 1979
Price: $50
Included with price: User notes and disk.
Author: Fred O. Kask
Where to purchase it:
Kask Labs
1207 E. Secretariat Drive
Tempe, AZ 85284

Program Name: Speedy Disk Copy
Hardware System: Mits Altair with 88-DCDD disk controller & two 8" hard sectored disk drives.
Minimum Memory Size: 32K RAM
Language: Altair Disk Basic (Rev. 3.4, 4.0, 4.1, & 5.0)
Description: The Speedy Disk Copy routine will copy all or any part of an Altair Disk Basic or DOS formatted diskette in less than 100 seconds. By comparison, the Altair supplied PIP utility requires about 40 minutes. The program is self-prompting, performs both read and write validation checks with a listing of the location and quantity of errors upon completion. Recorded twice in Altair Disk Basic ASCII format on 8" hard sectored diskette.
Release: Currently
Price: $19.50 Postpaid.
Author: Joe Konrad
Where to purchase it:
Jack Compute
33 Plant Street
New London, CT 06320

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: STAR*TRAC BASIC Debugger
Hardware System: North Star 5.1 or 5.2 DOS

Minimum Memory Size: 16K
Language: Assembler
Description: Extention to North Star Basic 5.1 offers the first fully interactive debug monitor for *any* microcomputer Basic. Allows user to insert breakpoint in Basic program and assume full keyboard control over subsequent execution. Upon reaching the breakpoint, program control is turned over to STAR*TRAC monitor, which allows execution of any direct mode command. Program variables can be examined or altered before resuming. The Basic program can then be single-stepped, with each program source line and value of selected variables displayed before execution. Single-step feature of STAR*TRAC extends to multiple commands on a source line: each individual command is executed separately. Breakpoint can be relocated anywhere within program, or invoked after a program command has been executed a specified number of times. Can assert a conditional breakpoint: control is assumed whenever a specified logical expression becomes true. Often a faulty program can only be identified by its results—the portion of the program responsible for the fault cannot be specified. The conditional breakpiont allows control over such a Basic program to be assumed when a specified program symptom occurs, such as when value of a variable is altered.
Release: 1980
Price: $49.00
Included with price: Basic modification; complete documentation is included and full user support is provided.
Author: Allen Ashley
Where to purchase it:
395 Sierra Madre Villa
Pasadena, CA 91107

Program Name: TAPEDISK, DISKTAPE, MFDT
Hardware System: CP/M and Processor Technology SOL or CUTS cassette I/O and SOLOS or CUTER monitor program.
Minimum Memory Size: 16K CP/M (about 30K for MFDT)
Language: 8080 assembly except MFDT is compiled from C.
Description: CP/M file distribution via cassette tapes. Transfer and sizes and types of CP/M files to and from CUTS format cassette tapes. Allows trading between systems with different disk types and provodes archival storage.
DISKTAPE writes one file to tape.
TAPEDISK reads entire tape to disk.
MFDT is optional but allows unattended writing of tapes from a list of ambiguous file names with spooling of console input and output to/from disk.
Release: Already in the field.
Price: $10 ($20 with MFDT).
Included with price: COM, DOC and source files on CUTS cassette with paper instructions to make tape load itself. Or send Micropolis Mod II diskette.
Where to purchase:
Richard Greenlaw
251 Colony Ct.
Gahanna, Ohio 43230

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.
Release: 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: Tarbell Dual-Density DMA Support Package
Hardware System: 8080/Z80 S100 system with Tarbell DD/DMA disk controller.
Minimum Memory Size: N/A
Language: 8080 Assembly Language (ASM or MAC)
Description: CP/M 2.0 compatible BOOT and BIOS for Tarbell Dual Density disk controller, including all support programs required for normal operation (FORMAT, Disk validation, Fast absolute copy, auto-density sysgen, etc.) Not compatible with public domain code from Tarbell, this is all new code which supports IBM standard gaps and header information, has no known bugs, and is very clearly written. Currently supports CP/M with 128 byte sectors only, but will allow user to format and validate diskettes in any of the following formats (sectors/track x bytes/sector): Single: 26 x 128, 13 x 256, 8 x 512, 4 x 1024 Double: 48 x 128, 26 x 256, 15 x 512, 8 x 1024 Supports standard IOBYTE, remote console auto answer dial-in access, etc. Console/printer I/O currently uses IMSAI SIO2-2 (very easy to modify).
Release: Available now
Price: $50.00
Included with price: 8" 3740 CP/M style disk with source for BOOT, BIOS, FORMAT, VALDSK, ADCOPY and SYSGEN. Note: CP/M 2.0 from Digital Research required.
Authors: Lawrence E. Highes and Sam H. Adams
Where to purchase it:
Mycroft Labs
P.O. Box 6045
Tallahassee, Fla 32301

Program Name: TCS Business Accounting Package
Hardware System: Any system using Microsoft Basic, CP/M
Minimum Memory Size: 48K
Language: Microsoft Basic
Description: A fully integrated business software arsenal including General Ledger (provides immediate financial information for your company by keeping thorough records of all financial transactions); Accounts Payable (maintains complete vendor/voucher history including check writing capabilities); Accounts Receivable (instant customer accounts information - current and aged - with complete invoicing and statement capabilities); Payroll (calculates payroll for every type employee while maintinaing monthly, quarterly and yearly totals for reporting purposes in multiple states. User modifiable tax tables. W-2, 941's, checks, etc.); Inventory Management (detailed inventory records, allows multiple item location and dept. ID, simplified posting and new easier-to-read reports.)
Release: GL,AP,AR,PR - 1978; IM - 1981
Price: Inventory: $400.00; GL,AP,AR,PR: $500.00 (for Microsoft Basic 4.5); IM: $400.00 (MBasic 5. x); GL,AP,AR,PR: $600.00 (for Microsoft Basic 5.X) GL,AP, AR, PR: $850.00 (for compiled version running on Microsoft Compiler.)
Included with price: Program disk, 600 page user manual, sample data and source code.
Author: TCS Software
Where to puchase it:
TCS Software
5582 Peachtree Road
Atlanta, GA 30341

Program Name: TED
Hardware System: 24K or larger Z80 CP/M system
Minimum Memory Size: 20K minimum, 24K recommended
Language: Z80 assembly
Description: TED is an advanced text editor which implements an enhanced subset of DEC TECO commands providing the following capabilities for editing of ASCII text.
*36 command/text buffers
*32 entry push down stack
*sophisticated macro command capability
*conditional and iterative command execution
*conditional and absolute branching
* multiple open files
Release: Available now
Price: $90.00
Included with price: 8" CPM compatible disk with object file, TED. COM and comprehensive manual (manual $20 if purchased separate).
Where to purchase it:
Small System Design
P.O. Box 4546
Manchester, New Hampshire 03108

Program Name: UDE-PRT
Hardware System: CP/M
Minimum Memory Size: 52K bytes
Language: Microsoft Basic

Description: Provides pagination and formatted file listings of Universal Data Entry (UDE) files. Batch and transaction totals are printed where defined. Optional date and report headings are provided.
Release: Dec. 1979
Price: $95; Licence Agreement Required
Included with price: Diskette, manual, examples, support
Author: The Software Store
Where to purchase it:
The Software Store
706 Chippewa Square
Marquette. MI 49855

Program Name: Utilities Software Disk (DMM-1)
Hardware System: CP/M 2.x or MP/M system
Minimum Memory Size: 16K
Language: Object Code
Description: Disk contains the following programs:

XDIR: Displays disk directory file names in alphabetic order and size of each file name. Also shown are the number of bytes on disk, number of file names in use, space used, number of available file names and space. Works on single and double density floppy disks as well as with hard disks.

EXTRACT: Will list a portion of a file between two label names. You do not have to list out whole file.

STRIP: Removes hex code from a PRN file and turns it back into an ASM file.

SORT: Creates symbol table from an assembly done with ASM that can be listed or used with Digital Research debugger SID.

CONVERT: Changes all uncommented lower case characters to upper case. Handy for nice looking listings and for assemblers that will not accept lower case.

STATUS: Provides system information such as memory available, TPA size, top of memory address, I/O assignments and more.

Release: September 1981
Price: $35 plus $1.50 shipping and handling
Included with price: Disk, 8" single density or 5-1/4" North Star
Where to purchase it:
Elliam Associates
2400 Bessemer St.
Woodland Hills, CA 91367
(213)348-4278

Program Name: VDRAW ASM
Hardware System: Any memory mapped video board with 2 x 3 Graphics: Polymorphic/IMSAI VIO/Vector G. Flashwriter
Minimum Memory Size: 1/4 K
Language: 8080 Assembler
Description: These routines will control memory-mapped video boards providing graphic capabilities. They will select and turn on or off any pixel desired. The user provides only an X and Y co-ordinate specifying the desired pixel for plot, or the X-Y co-ordinates of the start and end of a line. The routines will locate and set (or reset) the desired pixel or pixels. This will provide a simple interface for graphics from higher level languages. The plot routine will operate at very high speed. The draw routine, which utilizes the plot routine to set each pixel required, will draw a line on a video board so rapidly that the user will be unable to detect the time difference between the first and last pixels being set (or reset). The routine assumes that each pixel is controlled by a bit in an area occupied by a memory-mapped video board. The bits (pixels) must be arranged in a 2 x 3 matrix within a given byte (character) on the board. The two routines together will fill less than 256 bytes. The routines are also provided with two different methods for providing the X-Y addressing parameters. The parameters may be provided on the stack, or simply set into specified addresses.
Release: Currently available
Price: $30.00
Included with price: Program source code and documentation plus test program written in Basic.
Author: Hawkeye Grafix
Where to purchase it:
Hawkeye Grafix
23914 Mobile St.
Canoga Park, CA 91307

Program Name: VersaSort
Hardware System: CP/M
Minimum Memory Size: 32K
Language: 8080 machine code
Description: VersaSort will arrange any data files on the basis of key criteria; select up to 5 keys for each sort; and sort a file under the control of any CBasic program, quickly and easily.
Release: June 1980
Price: $195
Included with price: Documentation (75 pages) with many examples and samples.
Author: R. Murray
Where to purchase it:
MicroDaSys-Software
Box 36275
Los Angeles, CA 90036

Program Name: Video ASM
Hardware System: Any memory mapped video board
Minimum Memory Size: 1K
Language: 8080 Assembler
Description: This video driver presents the ultimate in flexibility. The driver can be rommed if the user desires. It requires about 3/4K, and fits easily in a 2708 EPROM. The program will drive any size video board, with any line width or number of lines, without revision. The configuration and address of the video board are parameters provided at run time. It is quite capable of driving several different video boards, or several different windows on the same board, simultaneously. All parameters are stored in an 18-byte area. To drive multiple displays simultaneously, the user need only switch in or out the 18-byte parameter table desired. All control characters are stored in a second table. These are moved from the program body to a second table area, so they are subject to execution time revision by the user, even when the driver resides in ROM. When used in conjunction with an IMSAI VIO or Vectorgraphic Flashwriter II, non-displayed 128 bytes of the VIO RAM to save all tables and variables. This driver offers such features as software scrolling, full cursor controls (up, down, left, right), screen clear, line erase, and user definable cursor character. It can be called with a single byte of data to be displayed, the address of a string to be displayed, or the address of a string to be displayed some variable number of times (repeat). The video driver will protect the contents of all registers during every call. They will be returned with their original contents.
Release: Currently available
Price: $40.00
Included with price: Program source code and documentation.
Author: Hawkeye Grafix
Where to purchase it:
Hawkeye Grafix
23914 Mobile St.
Canoga Park, CA 91307

Program Name: VSelect
Hardware System: PolyMorphic Systems 8813 single density
Minimum Memory Size: 8K
Language: 8080A Machine Language
Description: This program selects data file records. It is a general file utility program which searches an input data file of fixed length records for a specified character string. This program is an enhanced version of select which allows variable length fields within each data record. Use it to pick out all names beginning with a given letter, or to pick out everyone in a data list with a particular code. The output is versatile; a copy of the data record containing the match, or just its position in the file. You also have the choice of output to the screen, the printer or to create an output data file containing the output. The output files are compatible with Basic. Limited to 9999 records.
Release: September 1980
Price: $85
Included with price: Disk
Where to purchase it:
Ralph E. Kenyon Jr.
145-103 S. Budding Ave
Virginia Beach, VA 23452

Program Name: WHATSIT? (Wow! How'd All That Stuff get In There?) [WHATSIT? is a trademark of Computer Headware]
Hardware System: Any S-100 system; WHATSIT is available in Model NS-3 for North Star systems, and Model CP-2 for CP/M systems.
Minimum Memory Size: 32K (Model NS-3), 44K (Model CP-2).
Language: North Star BASIC (Model NS-3), CBASIC-2 (Model CP-2).
Description: WHATSIT is a self-indexing, cross referencing data query system. The program stores, indexes, and fetches free-format information in response to conversational "Requests." Typical queries range from "When's Johnny's Dental Checkup?" to "What's the U.N. Ambassador's Voting

Record?" WHATSIT's unique open-ended data structure evolves continuously during normal use, without respecifying the file. Unexpected new file headings are immediately added when first mentioned in a Request, then remain available for future reference. Always spoken of as "her" in the 160-page user's manual, WHATSIT distinguishes herself by her breezy, impertinent repartee, including such rejoinders as "News to me!" when queried for information not currently on file, or "Never mind!" when the operator cancels a Request unexpectedly.
Release: March 1978 (Model NS-3), August 1979 (Model CP-2).
Price: $125.00 (Model NS-3), $175.00 (Model CP-2).
Included with price: Disk with 160-page spiral bound user's manual.
Author: Computer Headware, Box 14694, San Francisco, CA 94114.
Where to purchase it:

Hardhat Software
Box 14815
San Francisco, CA 94114

Program Name: Wiremaster
Hardware System: Any Z-80 CP/M system
Minimum Memory Size: 48 Kbytes
Language: Written in ZSPL (Pete Ridley Software)
Description: Wiremaster is a software tool to aid in the design, layout, and construction of electronic hardware. Its inputs are easily derived from the schematic diagram and fed to Wiremaster in a CP/M text file. Outputs include a network map graphically showing all pins and wires, a wirelist sorted by lengths and levels, a parts list, and check lists that detect all wiring errors. The resulting information is then used for layout, error checking, wiring, component stuffing, and system debugging. Together with the schematic, this forms a complete and easily updated documentation package for an electronic product, and results in substantial savings of time.
Release: November 1980
Price: $75; manual $4
Authors: Jim and Gary Gilbreath
Where to purchase it:

Afterthought Engineering
7266 Courtney Dr.
San Diego, CA 92111

Program Name: WORD-C1, a text formatter
Hardware System: CP/M 2.2, an 8" dual diskette
Minimum Memory Size: 60K
Language: Compiled
Description: A text formatter to prepare letters, memos, reports, manuals, documents and books. Commands set page length, line width, skip pages and text, indent, center text, etc. Line spacing, filing, adjusting, margin right justification and page numbering are automatically controlled. WORD has no limit to text size, no special hardware or modification. The Mail merge option lets you merge text with data files created by IDM-C1.
Released: September 1981
Price: $85
Included with price: 8" diskette, user's manual & postage.
Where to purchase it:

Micro Architect Inc.
96 Dothan Street
Arlington, MA 02174
(617)643-4713

Program Name: Z-80 DES
Hardware System: Z-80 based
Minimum Memory Size: 16K
Language: Z-80 assembler
Description: High Speed Implementation of the NBS data encrypton algorithm. Modular and user oriented. Fully documented source code supplied for the algorithm. Special run-time package for TRS-80. Database protection, password scrambling, telecommunications security from remotely-accessed data files. Protect sensitive or proprietary files. Easily adapted by user for custom uses.
Release: April 1979
Price: $34.95 + $9.95
Included with price: Documented source code listing
Author: ITM
Where to purchase it:
Interface Technology of Maryland
P.O. Box 745
College Park, MD 20740

Program Name: Z-80 Floppy Disk Test
Hardware System: CP/M 2.0
Minimum Memory Size: 32 kbytes
Language: Z-80 Assembler
Description: An extremely fast, general purpose utility to test or initialize a diskette. When the program is loaded, the operator is asked a series of questions to define the test mode. Selectable options include: lock on read or write, restore original diskette data, fixed or semi-random data patterns, lock on track, lock on sector, error listings on console or printer. The program is supplied to test a standard single density soft sectored diskette, but allows the user to specify the number of tracks or sectors per track for other types of disk drives.
Release: Currently available
Price: $25.00
Included with price: Eight inch soft-sectored single-density diskette, detailed printed instructions.
Where to purchase it:

Laboratory Microsystems
4147 Beethoven Street
Los Angeles, CA 90066

Program Name: Z-80 FORTH
Hardware System: CP/M 2.0 or MP/M 1.0
Minimum Memory Size: 32K, may be reconfigured by user to take advantage of larger memory sizes.
Language: Z-80 Assembler
Description: Optimized fig-FORTH for Z-80 microcomputers. Uses standard CP/M random access disk files for screen storage. Extensions allow use of all CP/M functions. Distribution diskette includes: interpreter, line editor, screen editor, decompiler, debugging aids, utilities, several demonstration programs and over 100 Kbytes of documentation. Source code provided to extend vocabulary to meet FORTH-79 Standard.
Release: Currently available
Price: $50.00
Included with price: Eight inch soft-sectored, single-density diskette, 55 page user manual
Where to purchase it:

Laboratory Microsystems
4147 Beethoven Street
Los Angeles, CA 90066

Program Name: ZAS Z-8000 Development Package
Hardware System: Any 8080/Z80 standard CP/M system
Minimum Memory Size: 48K
Language: 8080 Machine Code
Description: ZAS is an assembly language development tool for Zilog's Z8001 and Z8002 16-bit microprocessors. Includes a relocatable cross-assembler, a linker/task builder, an absolute object file loader, and a Z-8000 run-time module, ZEX, which supports any Z-8000 alternate bus master (such as the Ithaca Intersystems MPU-8000). Using CP/M, ZEX creates an I/O-independent run-time environment for application code written with ZAS. The package provides a fully integrated software development environment for the Z-8000, while retaining full use of current software and hardware facilities under CP/M.
Release: March 1981
Price: $395, $25 for user manual
Included with price: ZAS Assembler, ZLK Task Builder, ZLD Object Loader, ZEX Run-Time Monitor, User Manual. (8" SD CP/M Format Floppy)
Where to purchase it:

Western Wares
P.O. Box 48
Placerville, CO 81430
(303)728-4266

Program Name: ZDM
Hardware System: CP/M
Minimum Memory Size: Overlays CCP
Language: Z-80 Machine Language
Description: ZDM is a powerful Z-80 debugger and monitor designed to operate as a replacement for DDT on any CP/M system. All DDT commands (except A) are implemented. Additionally, ZDM features: a) customization to user terminal specifications, b) display and/or alteration of either set of Z-80 registers and flags, c) enable or disable interrupts when entering target program and d) read from an input port or write to an output port. ZDM uses extended 8080 mnemonics similar to those for the Digital Research macroassembler, MAC. Available on 8-inch single density IBM disk, 5-inch single density North Star, or double density Micropolis.
Release: August 1980
Price: $30; Manual $3
Included with price: Disk, manual and copying instructions.
Where to purchase it:

RD Software
1290 Monument St.
Pacific Palisades, CA 90272

CP/M Programmer's Reference Guide

Sol Libes

BUILT-IN COMMANDS

```
DIR                     Display file directory {current drive
DIR d:                                         {designated drive
DIR filename.typ        Search for named file, current drive
DIR *.typ               Display all files of named type,curr drv
DIR filename.*          Display all types of designated filename
DIR x????.*             Display all filenames 5 characters
                          long and start with letter x

TYPE filename.typ       Display ASCII file {current drive
TYPE d:filename.type                       {designated drive

ERA filename.typ                 {named file, current drive
ERA *.*                          {all files, curr drv, V2.x curr user
ERA *.typ               Erase    {all files }designated{drive
ERA d:filename.typ               {named file}          {types
ERA filename.*                   {all types of named file, curr drv

REN nuname.typ=oname.typ  {REName file {current drive
REN d:nuname.typ=oname.typ}            {designated drive

SAVE n filename.typ     SAVE as named file {current drive
SAVE n d:filename.typ                      {designated drive
                        n pages (page=256 bytes) start @ 100H

d:                      Switch to designated disk drive
                          A-D V1.4; A-P V2.x
USER n                  Change user area (Version 2.x)
```

TRANSIENT COMMANDS

```
DDT                     Initiate Dynamic Debugger Tool program
DDT filename.typ        Initiate DDT and load named file

ASM filename            Assemble named ASM {current drive
ASM d:filename             file on:        {designated drive
ASM filename,abc        a=source file drv; b=HEX file destin-
                        ation drv (Z=skip); c=PRN file destin-
                        ation drv (X=console,Z=skip)
LOAD filename           Make .COM file from {current drive
LOAD D:filename           named HEX file on:{designated drive

DUMP filename.typ       Display file in hex {current drive
DUMP d:filename.typ                         {designated drive

MOVCPM n                        {and execute nKbyte CP/M system
MOVCPM n *              Create  {image of nKbyte CP/M system
MOVCPM * *                      {image of maxKbyte CP/M for
                                  SYSGEN or SAVE
SYSGEN                  Initiate SYStem GENerate program

SUBMIT filename parameters Execute SUB file using optional
                                parameter(s)
XSUB                    Execute eXtended SUBmit program (V2.x)

ED filename.typ         Execute EDitor program to create
ED d:filename.typ                 or edit named file

STAT                    Display STATus-R/W or R/O {current drv
STAT d:                   and available disk space {named drive
STAT DEV:                       {DEVice assignments
STAT VAL:                       {VALid device assignments
STAT DSK:               Display {DISK characteristics}
STAT USR:                       {current USeR areas  } V2.x
STAT filename.typ $S            {size of file        }
STAT filename.typ               {file characteristics{curr drv
STAT d:filename.typ                                  {named drv
STAT d:=R/O                     {designated drive to Read-only
STAT filename.typ $R/O                  {Read-only  }
STAT filename.typ $R/W Change           {Read-Write } V2.x
STAT filename.COM $SYS          {named file to {System file}
STAT filename.COM $DIR         {               {Drctry file}
STAT gd:=pd:            Change general device (CON:,LST:,PUN:
                          and/or RDR:) assignment of
                          physical device (see IOBYTE)
```

PIP COMMANDS

```
PIP                     Initiate Peripheral Interchange Program
*d:=s:filename.typ        Copy named file      {from source drv
*d:nuname.*=s:olname.typ Copy&change filename {to destinat drv
PIP d:=s:filename.typ    Initiate PIP and copy  named file

PIP d:=s:*.*              from source drv {all files
PIP d:=s:filename.*             to        {all named files
PIP d:=s:*.typ            destination drv {all files named typ
PIP LST:=filename.typ                     {list device
PIP PUN:=filename.typ     send named file to {punch device
PIP CON:=filename.typ                        {console device
PIP filename.typ=RDR:     Copy data from reader device to
                            named file (current drive)
*nuname.typ=aname.typ,bname.type,cnametyp {ASCII   }copy&con-
*d:nuname.type=s:aname.typ,s:bname.typ     {       }catenate
*nuname.typ=aname.typ[X],bname.typ[X]      {non-ASCII} files
PIP LST:=aname.typ,bname.typ        send files in sequence
PIP LST:=a:name.typ,s:name.typ       to list device
```

PIP PARAMETERS

```
          *filename.typ=RDR:[$]

[$]  - read data block until ^S character
[Dn] - delete characters past column n
[E]  - echo all copy operations to console
[F]  - remove form feeds
[Gn] - get file from n user area - V2.x
[H]  - check for proper hex format
[I]  - same as H plus ignores ":00"
[L]  - change all upper case characters to lower case
[N]  - add line numbers with leading zeros suppressed
[N2] - same as N plus no leading zeros
[O]  - object file transfer;ignores end-of-file
[P]  - "{insert form feed every {60}{lines
[Pn] - {                     {n }
[Qstring^Z] - Quit copying  after{ string is found
[Sstring^Z] - Start copying when {
[R]  - read SYS file (V2.x)
[Tn] - expand tab space to every n columns
[U]  - change all lower case characters to upper case
[V]  - verify copied data
[W]  - delete R/O files at destination (V2.x)
[X]  - copy non-ASCII files
[Z]  - zero parity bit on all characters in file
```

PIP KEYWORDS

```
CON: CONsole device (defined in BIOS)
EOF: send End-of-File (ASCII-^Z) to device
INP: INPut source (patched in PIP)
LST: LiST device (defined in BIOS)
NUL: send 40 NULLs to device
OUT: OUTput destination (patched in PIP)
PRN: same as LST:; tabs every 8th character, numbers
       lines & page ejects every 60 lines with
       initial eject
PUN: PUNch device   defined in BIOS
RDR: ReaDeR device

  refer to IOBYTE section for additional physical devices
```

COMMAND CONTROL CHARACTERS

charac	function	ASCII code
C	Reboot CP/M (warm boot)	03H
E	Start new line	05H
H	Backspace and delete (V2.x)	08H
I	Tab 8 columns	09H
J	Line feed	0AH
M	Carriage return	0DH
P	Printer on/printer off	10H
R	Retype current line ~	12H
S	Stop display output - any character except ^c restarts output	13H
U	Delete line	15H
X	same as ^U (V1.4) backspace to start of line (V2.x)	18H
Z	End of console input (ED & PIP)	1AH
delete	Delete and display	7FH
rubout	last character (tape only)	7FH

ASM CONVENTIONS

```
labels followed by colon   1- 6 alphanumeric characters
symbol (eq. EQU) no colon  first must be alpha, ? or .

Assembly Program Format (space seperates fields)
label:   opcode  operand(s)          ;comment

Operators (unsigned)
a+b      a added to b
a-b      difference between a and b
 +b      0+b (unary addition)
 -b      0-b (unary subtraction)
a*b      a multiplied by b
a/b      a divided by b (integer)
a MOD b  remainder after a/b
  NOT b  complement all b-bits
a AND b
a OR b   bit-by-bit (AND, OR, XOR) of a and b
a XOR b
a SHL b  shift a (left, right) b bits, and off, zero fill
a SHR b

Hierarchy Of Operations          Constants
 highest: * / MOD SHL SHR    Numeric (post radix)
          - +                    B=binary
          NOT                    O,Q=octal
          AND                    D=decimal(default)
  lowest: OR XOR                 H=Hexidecimal
                              ASCII - in quotes (e.g. 'A')

Pseudo-ops
ORG  const        Set program or data origin (default=0)
END  start        End program. Optional address where
                    execution begins
EQU  const        Define symbol value(may not be changed)
SET  const        Define symbol value(may be changed later)
IF const          Assemble block conditionally until ENDIF
ENDIF             Terminate conditional assembly block
DS const          Define storage space for later use
DB byte[,byte...,byte] Define bytes as numeric or ASCII
                    constants
DW word[,word...,word] Define word(s) (two bytes)

     const=constant (true if bit-0=1 otherwise false)
```

ASM ERROR CODES

```
D   Data error (element cannot be placed in data area)
E   Expression error (ill-formed expression)
L   Label error
N   Not implemented
O   Overflow (expression too complicated to compute)
P   Phase error (label has different values on each
      pass)
R   Register error (specified value not compatible
      with op code)
U   Undefined label (label) does not exist)
V   Value error (operand improper)
```

FILE TYPES

```
ASC  ASCII text file, usually Basic source
ASM  ASseMbly language file (source for ASM program)
BAK  BAcKup copy file (created by editor)
BAS  BASic source program file, usually tokenized
COM  COMmand file (transient executable program)
DAT  DATa file
DOC  DOCument file
FOR  FORtran source program file
INT  INTermediate Basic program file (executable)
HEX  HEXadecimal format file (for LOAD program)
LIB  Library file used by macro assembler
PLI  PL/I source file
PRN  PRINt file (source and object produced by ASM)
REL  RELocatable module
SAV  System file (V2.x)
SUB  SUBmit text file executed by SUBMIT program
SYM  SID symbol file
TEX  TEXt formatter source file
XRF  Cross reference file
$$$  Temporary file

Filename - 8 characters maximum
Filetype - 3 characters maximum

Invalid filename and filetype characters:
             < > . , ; : = ? [ ]
```

DDT COMMANDS

```
A sad            Assemble symbolic code ; start at sad

D                Dump RAM    (cad; 16 lines
D sad            to console  (sad; 16 lines
D sad,sad        from:       (sad thru sad

F sad,sad,const  Fill RAM from sad thru sad with constant

G                Start       (saved PC
G sad            program     (sad
G sad,bp1        execution   (sad and stop at bp1
G sad,bp1,bp2    at:         (sad and stop at bp1 or bp2
G,bp1,bp2                    (cad and stop at bp1 or bp2

H a,b            Display hex a+b and a-b

I filename       Set up FCB  (user code
I filename.typ    (5CH) for: (R-command (HEX or COM file)

L                Disassemble (cad; 12 lines
L sad            RAM         (sad; 12 lines
L sad,sad        from:       (sad thru sad

M sad,sad,nad    Move RAM block from sad thru sad to nad

R                Read file specified by I command to RAM at
R offset           normal address + optional offset

S sad            Substitute into RAM starting at sad

T n              Execute n instructions (default=1) with
                   register dump (trace)
U n              Execute n instructions (default=1) with
                   register dump after last instruction
Xr               Examine/change registers or flags
X                Examine registers (flag reg:C=carry, Z=zero,
                   M=sign, E=parity, I=aux carry)

     cad=current address       sad=start address
     nad=new address           pad=end address
     ?=error, can mean: file cannot be opened,checksum error
        in HEX file or Assembler/Disassembler overlayed.
```

ED COMMANDS

```
nA      Append n lines to buffer (n=0 -use half of buffer)
B       Move pointer to (beginning of file
-B                      (end
nC                      (forward n characters
nD      Delete n characters forward
E       End edit, close file, return to CP/M
nFs     Find n-th occurrance of string 's'
H       end edit, move pointer to beginning of file
I       Insert text at pointer until ^Z typed
Is      Insert string at pointer
nK      Kill n lines starting at pointer
nL      Move pointer n lines
nMx     execute command string 'x' n times
nNs     global F-command- until end of file
O       Abort ED, start over with original file
nP      List next n pages of 23 lines (n=0 -current page)
Q       Quit without changing input file
Rfn     Read fn.LIB into buffer at current pointer
n^x^y   Substitute string 'y' for next n forward
          occurrances of string 'x'
nT      Type n lines
U       change lower case to upper case (next entry)
V       enable internal line number generation
nW      Write n lines to output file (start at
          beginning of buffer)
nX      Write next n lines to file 'X$$$$$$.LIB'
nZ      Pause n/2 seconds (2MHz)
n       Move (forward (n lines)
<CR>         (forward (1 line ) and type one line
-            (backward
nrx     move to n line number and perform 'x' command
:mx     perform command 'x' from current line to line m
n:mx    move to n line number and perform command 'x'
        through line number m

note: "-" valid on all positioning and display commands
        for backward movement (e.g. -nC)
```

IOBYTE (0003H)

Device	LST:	PUN:	RDR:	CON:
Bit Position	7 6	5 4	3 2	2 1
Dec	Binary			
0	00	TTY:	TTY:	TTY:	TTY:
1	01	CRT:	PTP:	PTR:	CRT:
2	10	LPT:	UP1:	UR1:	BAT:
3	11	UL1:	UP2:	UR2:	UC1:

```
TTY: TeleTYpe
CRT: Cathode Ray Tube type terminal
BAT: BATch process(RDR=input,LST=output)
UC1: User defined Console
LPT: Line Printer
UL1: User defined List device
PTR: Paper Tape Reader
UR1: User defined
UR2: Reader devices
PTP: Paper Tape Punch
UP1: User defined Punch
UP2:   devices
```

low nibble = current drive (0=A,1=B,etc.)
high nibble = current user (V2.x only)

BIOS ENTRY POINTS

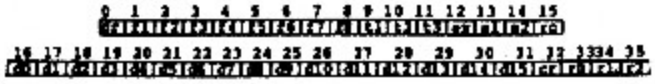
Hex addr	Vector Name	Function	Value Passed	Value Returned
**00	BOOT	cold start entry point		C=0
**03	WBOOT	warm start entry point		C=drv no
**06	CONST	check for console ready		A=const
**09	CONIN	read from console		A=char
**0C	CONOUT	write to console	C=char	
**0F	LIST	write to list device	C=char	
**12	PUNCH	write to punch device	C=char	
**15	READER	read from reader device		A=char
**18	HOME	move head to track-0		
**1B	SELDSK	select drive	C=drv no	HL=dph*
**1E	SETTRK	set track number	C=trk no	
**21	SETSEC	set sector number	C=sec no	
**24	SETDMA	set DMA address	BC=DMA	
**27	READ	read selected sector		A=dskst
**2A	WRITE	write selected sector		A=dskst
**2D*	LISTST	get list status		A=lstst
**30*	SECTRAN	sector translate subroutine	BC=lsecno DE=smap	HL=psysec

const=console status
00=idle
FF=data avail
dph=disk parameter/header address
dskst=disk status
00=OK
01=error
lstst=list status
00=busy
FF=ready

lsecno=logical sector number
psysec=physical sector number
smap=sector interlace map address
char=character
drv no=drive number
trk no=track number
sec no=sector number
DMA=DMA address
* not used in V1.4
** = contents of location 0002H

FILE CONTROL BLOCK

Byte[s]		function
0	dr	Drive code (0=current, 1=A, 2=B, etc)
1-8	f1-f8	File Name
9-11	t1-3	File Type t1=1-R/O; t2=1-SYS
12	ex	current EXtent number
13	s1	reserved (V1.4)
14	s2	-0 on BDOS call to Open,Make,search (V1.4 not used always 00H)
15	rc	extent Record Count
16-31	d0-dn	Disk map
32	cr	current record for r/w
33-35	rn	random record number



MEMORY ALLOCATIONS

(b=memsize-20K V2.x; memsize-16K V1.4)

	Hex Memory Locations	Contents
System Scratch Area (0-FFH)	0-2	jump to BIOS warm start entry point
	3	IOBYTE
	4	login drive number and current user
	5-7	jump to BDOS
	8-37	reserved: interrupt vectors & future use
	38-3A	RST7-used by DDT or SID programs
	3B-3F	reserved for interrupt vector
	40-4F	scratch area used by CBIOS
	50-5B	not used
	5C-7C	File Control Block (FCB) area (default)
	7D-7F	Random record position-V2.x (default)
	80-FF	DMA buffer area (128 bytes) for input and output (default)
Transient Program Area	100...33FF+b / 100...28FF+b	COM file area (V2.x / V1.4)
CCP area	3400+b-38FF+b / 2900+b-30FF+b	Console Command Processor (V2.x / V1.4)
BDOS area	3C00+b-49FF+b / 3100+b-3DFF+b	Disk Operating System (V2.x / V1.4)
BIOS area	4A00+b-4FFF+b / 3E00+b-3FFF+b	I/O system (V2.x / V1.4)

CP/M DISK FORMAT

Media: 8" soft-sectored floppy-disk single density (IBM 3740 standard)
Tracks: 77 (numbered 0 thru 76)
Sectors/Track: 26 (numbered 1 thru 26)
Bytes/Sector: 128 data bytes (one logical record)
Storage/Disk: 256,256 bytes (77*26*128)
File Size: any number of sectors from zero to capacity of disk.
Extent: 1Kbytes-8 sectors (smallest file space allocated)
Skew: 6 sectors standard (space between consecutive physical sectors on track): 1-7-13-19-25-5-11-17-23-3-9-15-21-2-8-14-20-26-6-12-18-24-4-10-16-22

System: Track 0 & 1 (optional)
Track-0,sector 1: boot loader
Track-0,sectors 2-26: CCP & BDOS
Track-1,sectors 1-17:
Track-1,sectors 18-26: CBIOS

Directory: Track 2: 16 sectors typ. 32-bytes/entry (64 entries typ.) - extents=0 and 1

User File Area: Remaining sectors on Track-2 and -3 to 76
Extents 2 and above

BDOS FUNCTION CALLS

(request to BDOS to perform specified functions)

	Function Number in C reg Dec	Hex	Function	Value Passed to BDOS in DE(or E)reg	Value Returned in A (or HL) reg
	0	00	system reset	--	--
	1	01	console read	--	char
	2	02	console write	E=char	--
	3	03	reader read	--	char
	4	04	punch write	E=char	--
Peripheral I/O	5	05	list write		--
	6	06	direct con IO (V1.x)	E=FFH(input) / char(output)	0=not ready / char
	7	07	get IOBYTE	--	IOBYTE
	8	08	set IOBYTE	E=IOBYTE	--
	9	09	print string	string addr	--
	10	0A	read console buffer	addr of data buffer	chars in buffer
	11	0B	get console status	--	00(not ready) FF(ready)
	12	0C	lift head(V1.x) get vers (V2.x)	-- --	-- HL=version no.
	13	0D	reset disk **	--	--
	14	0E	select disk	E=drive no	--
	15	0F	open file	FCB addr	dir FF(not found)
	16	10	close file		
	17	11	search for fil		
	18	12	search for next	--	* 00(valid)
Disk I/O	19	13	delete file		
	20	14	read next record	FCB addr	
	21	15	write next rec		
	22	16	create file		dir FF(disk full)
	23	17	rename file	old file FCB addr	directory code FF(not found)
	24	18	get login vectr	-- (V1.4)	HL=drive code
	25	19	get disk no.	--	A=cdn
	26	1A	set DMA addr.	DMA addr	--
	27	1B	get alloc vectr	--	HL=ava
	28	1C	write protect	--	--
	29	1D	get R/O vector	--	HL=R/O vector
	30	1E	set file attrib	FCB addr	dir
V2.x only	31	1F	get addr (disk parameters)	--	HL=dpba
	32	20	set/get user code	E= FFH(get) user code(set)	current code --
	33	21	read random	FCB addr	error code***
	34	22	write random		
	35	23	compute file size	(r0,r1,r2 format)	random record field set
	36	24	set random rec		
V2.2 & later	37	25	reset drive	drive vector	0
	40	28	write random with zero fill	FCB addr	return code
not used	38	26			
	39	27			

* V1.4 none
** V1.4 intializes system and selects A drive
*** error codes: 01-reading unwritten data
02-cannot close current extent
04-seek to unwritten extent
05-directory overflow (write only)
06-seek past physical end of disk

char=character (ASCII)
addr=address
dir =directory code
cdn =current drive number (A=0,B=1,etc)
dpba=disk parameter block address

CP/M Programmer's Reference Guide

Sol Libes

BUILT-IN COMMANDS

```
DIR                     Display file directory {current drive
DIR d:                                         {designated drive
DIR filename.typ        Search for named file, current drive
DIR *.typ               Display all files of named type,curr drv
DIR filename.*          Display all types of designated filename
DIR s????.*             Display all filenames 5 characters
                          long and start with letter s
TYPE filename.typ       Display ASCII file {current drive
TYPE d:filename.type                       {designated drive

ERA filename.typ          {named file, current drive
ERA *.*                   {all files, curr drv, V2.x curr user
ERA *.typ          Erase  {all files {designated {drive
ERA d:filename.typ        {named file}           {types
ERA filename.*            {all types of named file, curr drv

REN nuname.typ=oldname.typ  {REName file {current drive
REN d:nuname.typ=oldname.typ}            {designated drive

SAVE n filename.typ     SAVE as named file {current drive
SAVE n d:filename.typ                      {designated drive
                        n pages (page=256 bytes) start @ 100H

d:                      Switch to designated disk drive
                          A-D V1.4; A-P V2.x
USER n                  Change user area (Version 2.x)
```

TRANSIENT COMMANDS

```
DDT                     Initiate Dynamic Debugger Tool program
DDT filename.typ        Initiate DDT and load named file

ASM filename            Assemble named ASM {current drive
ASM d:filename             file on:        {designated drive
ASM filename,abc         a=source file drv; b=HEX file destin-
                           ation drv (Z=skip);c=PRN file destin-
                           ation drv (X=console,Z=skip)
LOAD filename           Make .COM file from {current drive
LOAD D:filename           named HEX file on:{designated drive

DUMP filename.typ       Display file in hex {current drive
DUMP d:filename.typ                         {designated drive

MOVCPM n                        {and execute nKbyte CP/M system
MOVCPM n *              Create  {image of nKbyte CP/M system
MOVCPM * *                      {image of maxKbyte CP/M for
                                          SYSGEN or SAVE
SYSGEN                  Initiate SYSteM GENerate program

SUBMIT filename parameters Execute SUB file using optional
                                parameter(s)
XSUB                    Execute eXtended SUBmit program (V2.x)

ED filename.typ         Execute EDitor program to create
ED d:filename.typ                 or edit named file

STAT                    Display STATus-R/W or R/O {current drv
STAT d:                   and available disk space {named drive
STAT DEV:                       {DEVice assignments
STAT VAL:                       {VALid device assignments
STAT DSK:               Display {DiSK characteristics}
STAT USR:                       {current USeR areas  } V2.x
STAT filename.typ $$            {size of file
STAT filename.typ               {file characteristics{curr drv
STAT d:filename.typ                                  {named drv
STAT d:=R/O                     {designated drive to Read-only
STAT filename.typ $R/O                      {Read-only  }
STAT filename.typ $R/W  Change  {named file to{Read-Write } V2.x
STAT filename.COM $$SYS                     {System file}
STAT filename.COM $DIR                      {Drctry file}
STAT gd:=pd:          Change general device (CON:,LST:,PUN:
                        and/or RDR:) assignment of
                        physical device (see IOBYTE)
```

PIP COMMANDS

```
PIP                     Initiate Peripheral Interchange Program
*d:=s:filename.typ        Copy named file       {from source drv
*d:nuname.*=s:oldname.typ Copy&change filename {to destinat drv
PIP d:=s:filename.typ     Initiate PIP and copy  named file
PIP d:=s:*.*            from source drv {all files
PIP d:=s:filename.*           to       {all named files
PIP d:=s:*.typ          destination drv {all files named typ
PIP LST:=filename.typ                      {list device
PIP PUN:=filename.typ   send named file to {punch device
PIP CON:=filename.typ                      {console device
PIP filename.typ=RDR:   Copy data from reader device to
                          named file (current drive)
*nuname.typ=aname.typ,bname.type,cnametyp {ASCII  }{copy&con-
*d:nuname.type=s:aname.typ,s:bname.typ     {       }{catenate
*nuname.typ=aname.typ[X],bname.typ[X]      {non-ASCII} files
PIP LST:=aname.typ,bname.typ        send files in sequence
PIP LST:=s:name.typ,s:name.typ       to list device
```

PIP PARAMETERS

```
          *filename.typ=RDR:[B]

[B]  - read data block until ^Z character
[Dn] - delete characters past column n
[E]  - echo all copy operations to console
[F]  - remove form feeds
[Gn] - get file from n user area - V2.x
[H]  - check for proper hex format
[I]  - same as H plus ignores ":00"
[L]  - change all upper case characters to lower case
[N]  - add line numbers with leading zeros suppressed
[N2] - same as N plus no leading zeros
[O]  - object file transfer;ignores end-of-file
[P]  -{insert form feed every {60}{lines
[Pn] -{                       {n }
[Qstring^Z] - Quit copying  after} string is found
[Sstring^Z] - Start copying when }
[R]  - read SYS file (V2.x)
[Tn] - expand tab space to every n columns
[U]  - change all lower case characters to upper case
[V]  - verify copied data
[W]  - delete R/O files at destination (V2.x)
[X]  - copy non-ASCII files
[Z]  - zero parity bit on all characters in file
```

PIP KEYWORDS

```
CON: CONsole device (defined in BIOS)
EOF: send End-of-File (ASCII-^Z) to device
INP: INPut source (patched in PIP)
LST: LIST device (defined in BIOS)
NUL: send 40 NULLs to device
OUT: OUTput destination (patched in PIP)
PRN: same as LST:; tabs every 8th character, numbers
       lines & page ejects every 60 lines with
       initial eject
PUN: PUNch device    defined in BIOS
RDR: ReaDeR device

  refer to IOBYTE section for additional physical devices
```

COMMAND CONTROL CHARACTERS

charac	function	ASCII code
C	Reboot CP/M (warm boot)	03H
E	Start new line	05H
H	Backspace and delete (V2.x)	08H
I	Tab 8 columns	09H
J	Line feed	0AH
M	Carriage return	0DH
P	Printer on/printer off	10H
R	Retype current line	12H
S	Stop display output - any character except ^C restarts output	13H
U	Delete line	15H
X	same as ^U (V1.4) backspace to start of line (V2.x)	18H
Z	End of console input (ED & PIP)	1AH
delete	Delete and display	7FH
rubout	last character (tape only)	7FH

ASM CONVENTIONS

```
labels followed by colon   1- 6 alphanumeric characters
symbol (eg. EQU) no colon  first must be alpha, ? or .

Assembly Program format (space separates fields)
label:   opcode  operand(s)          ;comment

Operators (unsigned)
a+b       a added to b
a-b       difference between a and b
 +b       0+b (unary addition)
 -b       0-b (unary subtraction)
a*b       a multiplied by b
a/b       a divided by b (integer)
a MOD b   remainder after a/b
  NOT b   complement all b-bits
a AND b                 (AND)
a OR b    bit-by-bit    (OR  ) of a and b
a XOR b                 (XOR)
a SRL b   shift a (left ) b bits, and off, zero fill
a SRR b           (right)

Hierarchy Of Operations          Constants
 highest: * / MOD SHL SHR    Numeric (post radix)
          - +                  B=binary
          NOT                  O,Q=octal
          AND                  D=decimal(default)
  lowest: OR XOR               H=Hexidecimal
                             ASCII - in quotes (e.g. 'A')

Pseudo-ops
ORG  const       Set program or data origin (default=0)
END  start       End program. Optional address where
                   execution begins
EQU  const       Define symbol value(may not be changed)
SET  const       Define symbol value(may be changed later)
IF const         Assemble block conditionally until ENDIF
ENDIF            Terminate conditional assembly block
DS const         Define storage space for later use
DB byte[,byte...,byte] Define bytes as numeric or ASCII
                   constants
DW word[,word...,word] Define word(s) (two bytes)

     const=constant (true if bit-0=1 otherwise false)
```

ASM ERROR CODES

```
D    Data error: (e)lement cannot be placed in data area)
E    Expression error (ill-formed expression)
L    Label error
N    Not implemented
O    Overflow (expression too complicated to compute)
P    Phase error (label) has different values on each
       pass)
R    Register error (specified value not compatible
       with op code)
U    Undefined label (label does not exist)
V    Value error (operand improper)
```

FILE TYPES

```
ASC  ASCII text file, usually Basic source
ASM  ASsembly language file (source for ASM program)
BAK  BAcKup copy file (created by editor)
BAS  BASic source program file, usually tokenized
COM  COMmand file (transient executable program)
DAT  DATa file
DOC  DOCument file
FOR  FORtran source program file
INT  INTermediate Basic program file (executable)
HEX  HEXadecimal format file (for LOAD program)
LIB  Library file used by macro assembler
PLI  PL/I source file
PRN  PRINt file (source and object produced by ASM)
REL  RELocatable module
SAV  System file (V2.x)
SUB  SUBmit text file executed by SUBMIT program
SYM  SID symbol file
TEX  TEXt formatter source file
XRF  Cross reference file
$$$  Temporary file

Filename - 8 characters maximum
Filetype - 3 characters maximum

Invalid filename and filetype characters:
             < > . , ; : = ? [ ]
```

DDT COMMANDS

```
A sad            Assemble symbolic code ; start at sad

D                Dump RAM    (cad; 16 lines
D sad            to console  (sad; 16 lines
D sad,ead        from:       (sad thru ead

F sad,ead,const  Fill RAM from sad thru ead with constant

G                Start       (saved PC
G sad            program     (sad
G sad,bp1        execution   (sad and stop at bp1
G sad,bp1,bp2    at:         (sad and stop at bp1 or bp2
G,bp1,bp2                    (cad and stop at bp1 or bp2

H a,b            Display hex a+b and a-b

I filename       Set up FCB  (user code
I filename.typ    (5CH) for: (R=command (HEX or COM file)

L                Disassemble (cad; 12 lines
L sad            RAM         (sad; 12 lines
L sad,ead        from:       (sad thru ead

M sad,ead,nad    Move RAM block from sad thru ead to nad

R                Read file specified by I command to RAM at
R offset           normal address + optional offset

S sad            Substitute into RAM starting at sad

T n              Execute n instructions (default=1) with
                   register dump (trace)
U n              Execute n instructions (default=1) with
                   register dump after last instruction
Xr               Examine/change registers or flags
X                Examine registers (flag reg:C=carry, Z=zero,
                   M=sign, E=perity, I=aux carry)

     cad=current address        sad=start address
     nad=new address            ead=end address
     ?=error, can mean: file cannot be opened,checksum error
        in HEX file or Assembler/Disassembler overlayed.
```

ED COMMANDS

```
nA       Append n lines to buffer (n=0 -use half of buffer)
B                    (beginning)
-B       Move pointer to (end      ) of file
nC                   (forward n characters
nD       Delete n characters forward
E        End edit, close file, return to CP/M
nFs      Find n-th occurrence of string 's'
H        end edit, move pointer to beginning of file
I        Insert text at pointer until ^Z typed
Is       Insert string at pointer
nK       Kill n lines starting at pointer
nL       Move pointer n lines
nMx      execute command string 'x' n times
nNs      global F-command- until end of file
O        abort ED, start over with original file
nP       List next n pages of 23 lines (n=0 -current page)
Q        Quit without changing input file
Rfn      Read fn.LIB into buffer at current pointer
nSx^Zy   Substitute string 'y' for next n forward
           occurrences of string 'x'
nT       Type n lines
U        change lower case to upper case (next entry)
V        enable internal line number generation
nW       Write n lines to output file (start at
           beginning of buffer)
nX       Write next n lines to file 'X$$$$$$.LIB'
nZ       Pause n/2 seconds (2MHz)
n                  (n lines)
<CR>     Move (forward ( 1 line ) and type new line
-                  (backward
n:x      move to n line number and perform 'x' command
:mx      perform command 'x' from current line to line m
n:mx     move to n line number and perform command 'x'
           through line number m

note: "-" valid on all positioning and display commands
        for backward movement (e.g. -nC)
```

IOBYTE (0003H)

Device		LST:	PUN:	RDR:	CON:
Bit Position		7 6	5 4	3 2	2 1
Dec	Binary				
0	00	TTY:	TTY:	TTY:	TTY:
1	01	CRT:	PTP:	PTR:	CRT:
2	10	LPT:	UP1:	UR1:	BAT:
3	11	UL1:	UP2:	UR2:	UC1:

```
TTY: TeleTYpe
CRT: Cathode Ray Tube type terminal
BAT: BATch process(RDR=input,LST=output)
UC1: User defined Console
LPT: Line Printer
UL1: User defined List device
PTR: Paper Tape Reader
UR1: User defined
UR2: Reader devices
PTP: Paper Tape Punch
UP1: User defined Punch
UP2:   devices
```

low nibble = current drive (0=A,1=B,etc.)
high nibble = current user (V2.x only)

BIOS ENTRY POINTS

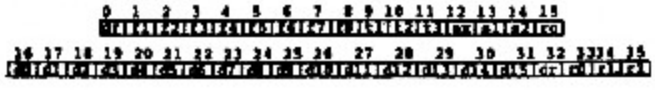
Hex addr	Vector Name	Function	Value Passed	Value Returned
**00	BOOT	cold start entry point		C=0
**03	WBOOT	warm start entry point		C=drv no
**06	CONST	check for console ready		A=const
**09	CONIN	read from console		A=chara
**0C	CONOUT	write to console	C=chara	
**0F	LIST	write to list device	C=chara	
**12	PUNCH	write to punch device	C=chara	
**15	READER	read from reader device		A=chara
**18	HOME	move head to track-0		
**1B	SELDSK	select drive	C=drv no	HL=dph*
**1E	SETTRK	set track number	C=trk no	
**21	SETSEC	set sector number	C=sec no	
**24	SETDMA	DMA address	BC=DMA	
**27	READ	read selected sector		A=dskst
**2A	WRITE	write selected sector		A=dskst
**2D*	LISTST	get list status		A=lstat
**30*	SECTRAN	sector translate subroutine	BC=lsecno DE=smap	HL=psysec

const=console status
00=idle
FF=data avail
dph=disk parameter/header address
dskst=disk status
00=OK
01=error
lstat=list status
00=busy
FF=ready

lsecno=logical sector number
psysec=physical sector number
smap=sector interlace map address
chara=character
drv no=drive number
trk no=track number
sec no=sector number
DMA=DMA address
* not used in V1.4
**= contents of location 0002H

FILE CONTROL BLOCK

Byte(s)		Function
0	dr	Drive code (0=current, 1=A, 2=B, etc)
1-8	f1-f8	File Name
9-11	t1-t3	File Type t1=1-R/O; t2=1-SYS
12	ex	current Extent number
13	s1	reserved (V1.4 not used)
14	s2	=0 on BDOS call to Open,Make,search (V1.4 always 80H)
15	rc	extent Record Count
16-31	d0-dn	Disk map
32	cr	current record for r/w
33-35	rn	random record number



MEMORY ALLOCATIONS

(b=memsize-20K V2.x; memsize-16K V1.4)

	Hex Memory Locations	Contents
	0-2	jump to BIOS warm start entry point
	3	IOBYTE
	4	login drive number and current user
System Scratch Area (0-FFH)	5-7	jump to BDOS
	8-37	reserved: interrupt vectors & future use
	38-3A	RST7=used by DDT or SID programs
	3B-3F	reserved for interrupt vector
	40-4F	scratch area used by CBIOS
	50-5B	not used
	5C-7C	File Control Block (FCB) area (default)
	7D-7F	Random record position-V2.x (default)
	80-FF	DMA buffer area (128 bytes) for input and output (default)
Transient Program Area	100...33FF+b / 100...28FF+b	COM file area V2.x / V1.4
CCP area	3400+b-38FF+b / 2900+b-30FF+b	Console Command Processor V2.x / V1.4
BDOS area	3C00+b-49FF+b / 3100+b-3DFF+b	Disk Operating System V2.x / V1.4
BIOS area	4A00+b-4FFF+b / 3E00+b-3FFF+b	I/O system V2.x / V1.4

CP/M DISK FORMAT

Media: 8" soft-sectored floppy-disk single density (IBM 3740 standard)
Tracks: 77 (numbered 0 thru 76)
Sectors/Track: 26 (numbered 1 thru 26)
Bytes/Sector: 128 data bytes (one logical record)
Storage/Disk: 256,256 bytes (77*26*128)
File Size: any number of sectors from zero to capacity of disk.
Extent: 1Kbytes-8 sectors (smallest file space allocated)
Skew: 6 sectors standard (space between consecutive physical sectors on track): 1-7-13-19-25-5-11-17-23-3-9-15-21-2-8-14-20-26-6-12-18-24-4-10-16-22

System: Track 0 & 1 (optional)
Track-0,sector 1: boot loader
Track-0,sectors 2-26; Track-1,sectors 1-17: CCP & BDOS
Track-1,sectors 18-26: CBIOS

Directory: Track 2; 16 sectors typ. 32-bytes/entry (64 entries typ.) - extents-0 and 1

User File Area: Remaining sectors on Track-2 and -3 to 76
Extents 2 and above

BDOS FUNCTION CALLS

(request to BDOS to perform specified functions)

	Function Number in C reg Dec	Hex	Function	Value Passed to BDOS in DE(or E)reqs	Value Returned in A (or HL) regs
	0	00	system reset	--	--
	1	01	console read	--	char
	2	02	console write	E=char	--
	3	03	reader read	--	char
	4	04	punch write	E=char	--
Peripheral I/O	5	05	list write		--
	6	06	direct con IO (V2.x)	E=FFH(input) / char(output)	0=not ready / char
	7	07	get IOBYTE	--	IOBYTE
	8	08	set IOBYTE	E=IOBYTE	--
	9	09	print string	string addr	--
	10	0A	read console buffer	addr of data buffer	chars in buffer
	11	0B	get console status	--	00(not ready) / FF(ready)
	12	0C	lift head(V1.x) / get vers (V2.x)	--	-- / HL=version no.
	13	0D	reset disk **	--	--
	14	0E	select disk	E=drive no	--
	15	0F	open file		dir
	16	10	close file	FCB addr	FF(not found)
	17	11	search for file		* / 00(valid)
	18	12	search for next	--	
Disk I/O	19	13	delete file		
	20	14	read next record	FCB addr	
	21	15	write next record		
	22	16	create file		dir / FF(disk full)
	23	17	rename file	old file FCB addr	directory code / FF(not found)
	24	18	get login vector	-- (V1.4)	HL=drive code
	25	19	get disk no.	--	A=cdn
	26	1A	set DMA addr.	DMA addr	--
	27	1B	get alloc vector	--	HL=avs
	28	1C	write protect	--	--
	29	1D	get R/O vector	--	HL=R/O vector
	30	1E	set file attrib	FCB addr	dir
V2.x only	31	1F	get addr (disk parameters)	--	HL=dpbe
	32	20	set/get user code	E= FFH(get) user code(set)	current code
	33	21	read random	FCB addr	error code***
	34	22	write random		
	35	23	compute file size	FCB addr (r0,r1,r2 format)	random record field set
	36	24	set random rec		
V2.2 & later	37	25	reset drive	drive vector	0
	40	28	write random with zero fill	FCB addr	return code
not used	38	26			
	39	27			

* V1.4 none
** V1.4 initializes system and selects A drive
*** error codes: 01-reading unwritten data
03-cannot close current extent
04-seek to unwritten extent
05-directory overflow (write only)
06-seek past physical end of disk

char=character (ASCII)
addr=address
dir =directory code
cdn =current drive number (A=0,B=1,etc)
dpba=disk parameter block address