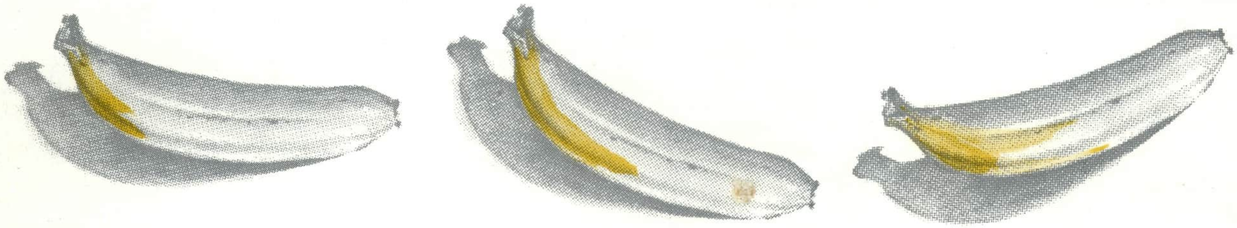
LIFELINES™
The Software Magazine™

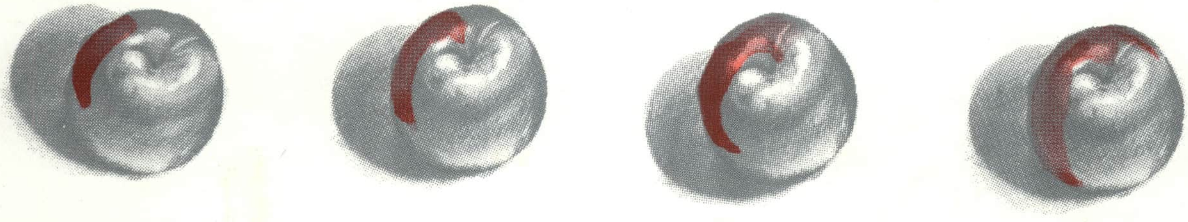
$3.00 November 1982 Volume III, No. 6 (ISSN 0279-2575, USPS 597-830)



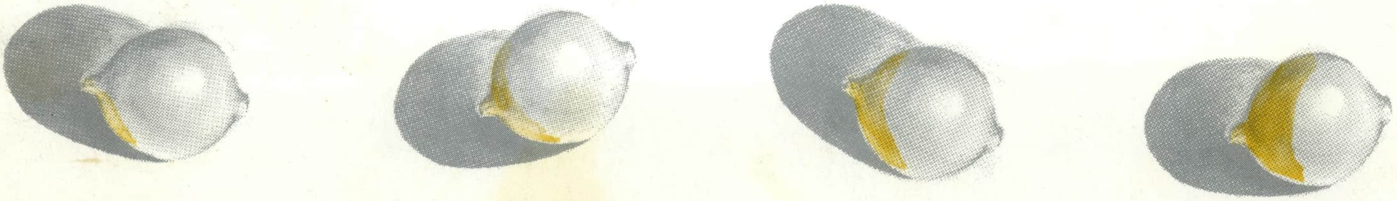
Apples From Oranges: Database Management

Reviews of DataFlex and Access Manager

CP/M's BDOS, Five New CPMUG Volumes



Improving on STAT, Tutorials, and More



TIM™ III

The Non-Programming Approach to Data Base Management

Data Base Management

Data management packages were created to save time and money in the development of software solutions to information problems. Many have been designed to accomplish just that, although most have only the programmer in mind. Sure they would save time in the long run, but what of the initial investment in time and effort required to learn the new language? What about the non-programmers in the world who would like an easy yet powerful applications generator? The solution is one of the most highly acclaimed software packages of our time, T.I.M. III.

What is T.I.M.?

T.I.M. is **Total Information Management.** Programmers love it due to its original solutions to classic data management problems. Non-programmers adore it since they can use it to achieve the same results as with other more complicated programming-like packages.

What Makes T.I.M. So Simple to Use?

We at Innovative Software, Inc. designed T.I.M. from day one with the end user in mind. Maybe he is a programmer who doesn't have time to learn a new language. Or perhaps a neophyte who fears coding pads and lines numbered by tens. We felt that a data management package should be able to be used by anyone from a systems analyst to a secretary. That's why T.I.M. takes a full *menu-driven* approach, uses multiple *HELP* screens, and has a manual that sets a new standard in documentation.

The Manual

Many people believe that the manual is just as important as the software itself, a view that we at Innovative Software, Inc. tend to share. The manual for T.I.M. is divided into two sections, the Reference section and the Primer. The Reference section describes all of T.I.M.'s commands and subcommands. This is done in English, not in technical terms or in our own language. Even if you have never seen a computer before in your life, you'll be able to read and understand our manual immediately. The second section is a primer which goes through several examples for you, again in plain English. These true-to-life examples take the beginner by the hand, and instructs him what to do and when. You will be able to see for yourself that T.I.M.'s only limitation is the imagination of the user.

Features of T.I.M.

T.I.M. has all of the features one has come to expect from a data management package, as well as many new ones. For example, a *word processing* interface that allows you to merge information from a T.I.M. file with letters or other documents created by a word processor. Now you can automatically send personalized letters to hundreds or thousands—quickly and easily. T.I.M.'s *Select* command enables you to pull specific information from a file. For example. "All customers who live in a certain ZIP code, whose last name begins with the letter A to L, whose balance due is less than $50.00." A sophisticated *report generator* and even a *list generator* are also included.

How powerful is T.I.M.? With a maximum record size of 2400 characters and the ability to keep up to forty fields sorted properly at all times, T.I.M. is powerful enough to handle just about any application. T.I.M. can handle over 32,000 records per file, and two files can be linked together for reports if your application requires a many-to-one relationship. T.I.M. also includes all of the same editing commands as your word processor, thus making data entry and editing a snap. You can also pull selected records from one file to place them into another. Files may be restructured to add or subtract fields and/or change field lengths or types. T.I.M. even has it's own utility for backing up hard disks onto floppies.

Where to Find T.I.M.

T.I.M. is available from **Lifeboat Associates.** Or you may purchase from us direct by calling 913/383-1089. Either way you will have the finest data management program available.

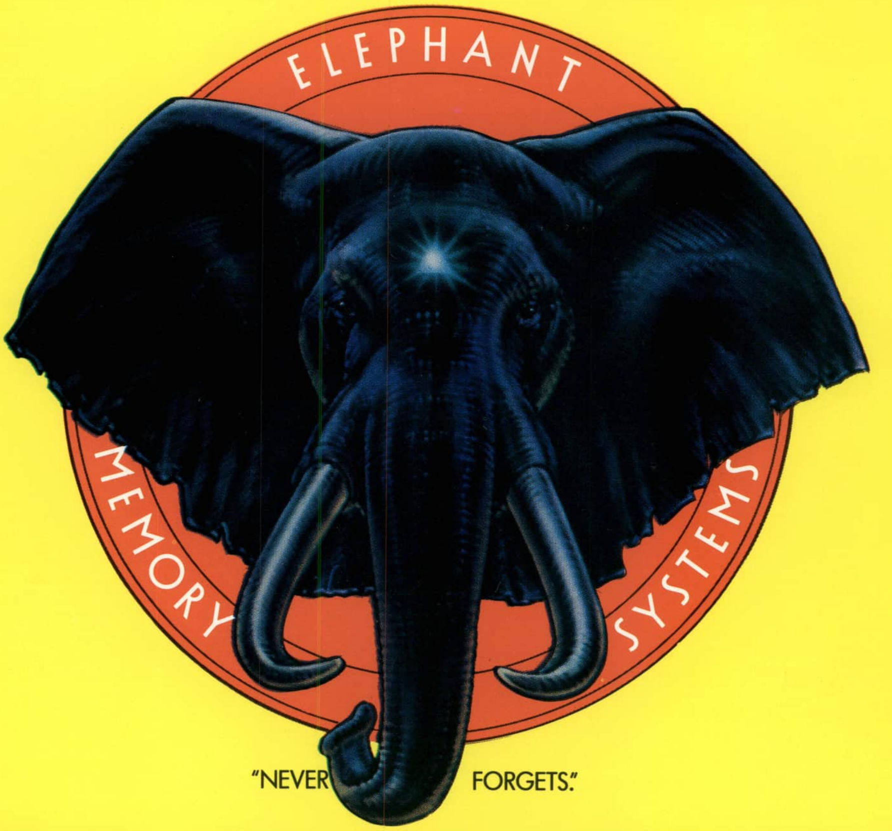
Innovative Software

Available for CP/M,* and IBM PC DOS.**
CP/M version—$695. IBM PC version—$495.

Innovative Software, Inc.
9300 W. 110th Street, Suite 380
Overland Park, Kansas 66210 USA
913/383-1089

TIM is a Trademark of Innovative Software, Inc.
*CP/M and MP/M are Trademarks of Digital Research
**Trademarks of IBM

REMEMBER:



MORE THAN JUST ANOTHER PRETTY FACE.

Says who? Says ANSI.

Specifically, subcommittee X3B8 of the American National Standards Institute (ANSI) says so. The fact is all Elephant™ floppies meet or exceed the specs required to meet or exceed all their standards.

But just who is "subcommittee X3B8" to issue such pronouncements?

They're a group of people representing a large, well-balanced cross section of disciplines—from academia, government agencies, and the computer industry. People from places like IBM, Hewlett-Packard, 3M, Lawrence Livermore Labs, The U.S. Department of Defense, Honeywell and The Association of Computer Programmers and Analysts. In short, it's a bunch of high-caliber nitpickers whose mission, it seems, in order to make better disks for consumers, is also to make life miserable for everyone in the disk-making business.

How? By gathering together periodically (often, one suspects, under the full moon) to concoct more and more rules to increase the quality of flexible disks. Their most recent rule book runs over 20 single-spaced pages—listing, and insisting upon—hundreds upon hundreds of standards a disk must meet in order to be blessed by ANSI. (And thereby be taken seriously by people who take disks seriously.)

In fact, if you'd like a copy of this formidable document, for free, just let us know and we'll send you one. Because once you know what it takes to make an Elephant for ANSI . . .

We think you'll want us to make some Elephants for you.

ELEPHANT.™ HEAVY DUTY DISKS.

For a free poster-size portrait of our powerful pachyderm, please write us.
Distributed Exclusively by **Leading Edge Products, Inc.,** 225 Turnpike Street, Canton, Massachusetts 02021
Call: toll-free 1-800-343-6833; or in Massachusetts call collect (617) 828-8150. Telex 951-624.

Yes, we're the biggest.

But that isn't what makes us number one.

It's the totality of what we do to make microcomputers more effective for you that makes us number one.

Yes. We have the largest number of packages—simple and complex. Yes. We have the greatest number of formats. Yes. We have the best technical support in the business. Yes. All of our products are immediately available.

But let's take a step back. When the microcomputer world opened up there was little definition and no software. Then came Lifeboat—to meet the need for easy-to-use, fully-tested, reliable software backed by extensive and available service.

Lifeboat developed standards for the industry which led to improved quality, reduced costs, higher levels of technical competence, credibility and reduced user risk.

Today Lifeboat offers personal, professional and corporate end-users, as well as dealers, distributors, authors, OEMs and others, a unique, single-source, full-service Software Support System.™

Everyone looks to us as the source of the most comprehensive, fully-tested line of software. Word processing, financial planning, accounting, graphics, data base management, languages and more. We have it all—for nearly every microcomputer available, including the IBM PC.

Our customer service department provides facilities for mail, telephone, TWX, telex and personal sales. We have a network of offices in the U.S., England, France, Japan, Switzerland and West Germany.

We provide a Software Desk Reference™ which contains up-to-date information about state-of-the-art software books, periodicals and accessories.

We offer subscriptions to *Lifelines*™ The Software Magazine.™ A monthly publication covering new products, tips for microcomputer users, product comparisons and other features to guide the reader before and after a purchase.

As the largest publisher of software, we also print a guide setting standards for software authors.

It takes a lot to become big but it takes even more to become—and remain—number one.

That's our commitment.

Lifeboat Associates, 1651 Third Avenue, New York, New York 10028.
TWX: 710-581-2524 (LBSOFT NYK). TELEX: 640693 (LBSOFT NYK).

□ Please send me a free Lifeboat Software Desk Reference filled with descriptions of over 200 programs designed for use in professional, business, programming and personal environments.

□ Please send further information on *Lifelines* and Author's Guide.

Name ________________ Title ________

Company ________________ Bus. Phone ________

Street ________________

City ________ State ________ Zip ________

Lifeboat Associates
World's No. 1 source of micro software

Lifelines, The Software Magazine,™ Lifelines Pub. Co.
Software Support System, Software Desk Reference,™ Lifeboat Assoc.

NEW 16-Bit Software Available for the IBM PC, plus . . .

System Tools:
Emulator/86
EM80/86
PMATE-86
UT86
PANEL-86

Telecommunications:
ASCOM

Languages:
Lattice C Compiler
PL/M

Word Processing Systems And Aids
WordStar
MailMerge
MicroSpell
Spellguard

Data Management Systems:
T.I.M. III

Mailing List Systems
Postmaster

Financial Accounting Packages
General Ledger

Numerical Problem-Solving Tools
Math PC
Plan86
SigmaCalc
Statpak

Professional And Office Aids
Dental Mngmnt Sys. (8000 & 9000)
Insurance Agency
Legal Time Acctng.
Medical Mngmnt Series
(8000 & 9000)

Disk Operating Systems:
MS-DOS (SB-86) — available for OEM license.

8-Bit Software Available

System Tools:
BUG and uBUG
DESPOOL
DISILOG
DISTEL
EDIT
EDIT-80
FILETRAN
IBM/CPM
MAC
MACRO-80
MINCE
PANEL
PASM
PLINK
PLINK II
PMATE
RAID
Reclaim
SID
TRS-80 Model II Cust. Disk
Unlock
WordMaster
XASM: 05, 09, 18, 48, 51, 65, 68, 75, F8, 400, Z8
ZAP80
ZDT
Z80 Development Package
ZSID

Telecommunications:
ASCOM
BSTAM
BSTMS
eZmail
MicroLink-80
RBTE-80

Languages:
ALGOL-60
APL/V80
BASIC Compiler
BASIC-80
baZic II
BD Software C Compiler
CBASIC-2
CIS COBOL (Standard)
COBOL-80
FORTRAN-80
KBASIC
JRT Pascal
muLISP/muSTAR
Nevada COBOL
Pascal/M
Pascal/MT
Pascal/M +
Pascal/Z
PL/I-80
Precision BASIC
STIFF UPPER LISP
S-BASIC
Timin FORTH
Tiny-C
Tiny-C TWO
UCSD Pascal
Whitesmiths' C Compiler
XYBASIC

Language and Applications Tools:
BASIC Utility Disk
DataStar
FABS
FABS II
Forms 2 for CIS COBOL
MAG/sam3,4
MAG/sort
M/SORT for COBOL 80
Programmer's Apprentice
PSORT
QSORT
STRING/80
STRING BIT
SuperSort
ULTRASORT II
VISAM

Word Processing Systems and Aids:
Benchmark
DocuMate/Plus
Letteright
MagicPrint
Magic Wand
Math★
MicroSpell
SMARTKEY
Spellguard
TEX
Textwriter III
WordIndex
WordStar
WordStar French
WordStar Customization Notes

Data Management Systems:
CONDOR
dBASE II
Formula
HDBS
Hoe
MAG/base1,2,3
MDBS
MicroSEED
T.I.M. III

General Purpose Applications:
CBS
CBS Label Option Pak
Selector III-C2
Selector IV

Mailing List Systems:
Benchmark Mailing List
Mailing Address
MailMerge for WordStar
NAD
Postmaster

Financial Accounting Packages:
BOSS Financial Accounting System
Financial Pkgs. (PTree)
Financial Pkgs. (SSG)
General Ledger Acctng (Univair)
GLector

Numerical Problem-Solving Tools:
Analyst
fpl
Microstat
muSIMP/muMATH
PLAN80
SigmaCalc
Statpak
T/MAKER II

Professional And Office Aids:
Apartment Mngmnt (Cornwall)
Datebook
Dental Mngmnt (Univair)
Dental Mngmnt-Family (Univair)
GrafTalk
Insurance Agency Mngmnt
Legal Time Acctng (Univair)
Medical Mngmnt (Univair)
Medical Mngmnt-Family (Univair)
PAS 3 Medical
PAS 3 Dental
Professional Time Acctng (PTA)
Property Mngmnt Pkg. (Am. Soft.)
Property Management (PTree)
Sales Pro
Wiremaster

Lifeboat After Hours
Backgammon/Gomoku

Educational Tools
Torricelli Author
Torricelli Studio

Books and Periodicals
APL—An Interactive Approach
Accounts Payable and Accounts Receivable-CBASIC
CBASIC User Guide
The Computer Glossary
The CP/M Handbook (with MP/M)
The C Programming Language
Crash Course in Microcomputing
Devil's DP Dictionary
Discover FORTH
DON'T (Or How To Care For Your Computer)
8080/Z80 Assembly Language Techniques For Improved Programming
Executive Computing
Fifty BASIC Exercises
General Ledger-CBASIC
Introduction to Pascal
Lifelines/The Software Magazine
Pascal User Manual and Report
The Pascal Handbook
The Pascal Primer
Payroll with Cost Accounting —CBASIC
Structured Microprocessor Programming
A User Guide To The UNIX System
Using CP/M—A Self-Teaching Guide

Hardware and Accessories
DC Data Cartridges
Diskette Drive Head Cleaning Kits
Flippy Disk Kit
Floppy Saver
Smartmodem
Vari Clean Cleaning Kit

Disk Operating Systems
BRIDOS
CP/M-80
MP/M
SB-80
APPLI-CARD
Softcard

Hard Disk Integration Modules

Media & Formats for 8-AND 16-Bit Microcomputers

This list of available formats is subject to change without notice. If you do not see your computer listed or are uncertain, call to confirm the format code for any particular equipment.

A.B. Dick	M8
ADDS Multivision	RT
AES Super Plus IV	Q4
ALSPA 8″	A1
Altair 8800	B1
Altos	A1
Apple CP/M-80 13 Sector	RG
Apple CP/M-80 16 Sector	RR
Archives 1	SG
AVL Eagle I	RB
AVL Eagle II	ST
BASF System 7100	RD
Blackhawk Micropolis Mod II	Q2
BMC iF-800	SR
Cado	A1
California Computer Sys 8″	A1
CDS Versatile 3B	Q1
CDS Versatile 4	Q2
Columbia Data Products 8″	A1
Columbia Data Products 5¼″	S4
Commodore CBM/PET + SSE Box + 8050	C2
Commodore CBM/PET w/Madison Z-RAM + 8050	C4
COMPAL-80	Q2
Compucorp 655	Q7
Compucorp 685	Q6
Computer Ops N.C. HQ	S2
Control Data 110	A1
CPT 8000	A1
Cromemco System 3	A1
Cromemco System 2 SD/SS	R6
Cromemco System 2 DD/SS	RX
Cromenco System 2 DD/DS	RY
CSSN Backup	T1
Datapoint 1550/2150 DD/SS	AA
Datapoint 1550/2150 DD/DS	AB
Datavue DU 80-222	M7
DEC VT 18 X	SD
Delta Systems	A1
Digi-Log Microterm II	RD
Digi-Log Sys. 1000/1500/2000	RD
Direct OA1000	M2
DTC Micro 210A	SC
Durango F-85	RL
Dynabyte DB8/2	R1
Dynabyte DB8/4	A1
Exidy Sorcerer + LB CP/M-80 5¼″	Q2
Exidy Sorcerer + Exidy CP/M-80 5¼″	RW
Exidy Sorcerer + Exidy CP/M-80 8″	A1
EXO	A1
Exxon 510/520	Q5
Findex	P6
Godbout	E1
Heath H8 + H47	A1
Heath H89 + Magnolia CP/M-80	P7
Heath H89 + Heath CP/M-80	P7
Helios II	B2
Heurikon MLZ, SS	SN
Heurikon MLZ, DS	SO
Heuristics HCC Spectrum	A1
Hewlett-Packard-87	SB
Hewlett-Packard 125, 5¼″	SB
Hewlett-Packard 125, 8″	A1
IBEX 7100	RQ
IBM Personal Computer	G1
ICL Personal Computer	RE
iCOM 2411 Micro Floppy	R3
iCOM 3712	A1
iCOM 3812	A1
iCOM 4511 Cartr. CP/M v.1.4	D1
iCOM 4511 Cartr. CP/M v.2.x	D2
IMSAI VDP-40/VDP-42	R4
IMSAI VDP-44	R5
IMSAI VDP-80	A1
Industrial Microsystems 5000	RA
Industrial Microsystems 8000	A1
Intel iPDS	M6
Intel MDS SD	A1
Intersil Development Sys	A1
Inter Systems Ithaca 800	A1
Intertec Superbrain DOS 0.5-2.x	RJ
Intertec Superbrain DOS 3.x	RK
Intertec Superbrain QD	RS
ISC Intecolor 8063/8360/8963	A1
Lanier EZ-1	M3
Lanier Super	Q4
Lexitron VT 1303 DS/DD	S8
Lexor Alphasprint Model S1	S1
Lexor Lexoriter	S1
Meca Delta-1 5¼″	P6
MICOM 2001	B3
MICOM 2001E	B4
MICOM 3003	M1
Micromation	A1
MicroMega 85	SC
Micropolis Mod 1	Q1
Micropolis Mod II	Q2
MITS 3200-3202	B1
Monroe OC 8820, DD/SS	SW
Morrow Discus	A1
Mostek	A1
MSD 5¼″	RC
MULTI-TECH-I	Q2
MULTI-TECH-II	Q2
Nascom (Gemini drives)	R3
Nascom II with Lucas Drives	SL
National MSC 6600	A1
NCR 8140/9010	A1
NEC PC-8001	RV
Nicolet Logic Analyzer Model 764	SX
NNC-80/80W	A1
North Star SD	P1
North Star DD	P2
North Star QD	P3
Northern Telecom 503	SM
Nylac Micropolis Mod II	Q2
Ohio Scientific C3	A3
OKI iF-800 + MSA CP/M-80	SP
OKI iF-800 + OKI/LB CP/M-80	SR
Osborne-1	SA
Otrona Attache	MC
Pertec PCC 2000	A1
PET/CBM + SSE Bx + 8050	C2
PET/CBM w/Madison Z-RAM + 8050	C4
Philips P-2000	MA
Philips MICOM 2001 8″	B3
Philips MICOM 2001E	B4
Philips MICOM 3003	M1
Processor Technology Helios II	B2
Quasar QDP100	A1
Quay 500	RQ
Quay 520	RP
Quay 900	A1
RAIR DD	RE
RAIR SD	R9
Research Machines 5.¼″	RH
Research Machines 8″	A1
Sanco 7000 5″	RQ
Sanyo MBC 1000	SY
Sanyo MBC 2000	SS
Sanyo MBC 3000	A1
Seattle	E1
Sony	U1
SD Systems 5¼″	R3
SD Systems 8″	A1
Spacebyte	A1
Tarbell 8″	A1
TecMar	E1
TEI 5¼″	R3
TEI 8″	A1
Televideo DD/DS	S5
T.I.P. (Alloy Engineering, Inc.)	T3
Toshiba T200	SF
Toshiba T250	A1
Triumph Adler Alphatronic	SV
TRS Model I + Omikron 5¼″	RM
TRS Model 1 + FEC Freedom	RN
TRS-80 Model 1 + Shuffleboard	A1
TRS-80 Model II	A1
Vector MZ	Q2
Vector System 2800	A1
Vector System B/VIP	Q2
Vista V-80 5¼″ SD	R8
Vista V200 5 DD	P6
Wangwriter	SE
WORDPLEX	SZ
XEROX 820, 5¼″	S6
XEROX 820, 860 8″	A1
ZEDA 580	SH
Zenith Z89 + Magnolia CP/M-80	P7
Zenith Z89 + Zenith CP/M-80	P7
Zenith DD/SS	SK
Zenith DD/DS	SJ
Zilog MC 22-20/25/50	A1

Program names and computer names are generally trademarks or service marks of the author or manufacturing company.

All Lifeboat (LB) 8-bit software requires SB-80 (or other CP/M-80 compatible disk operating system) unless otherwise stated.

All products are subject to terms and conditions of sale.

Copyright©1982 Lifeboat Associates

LIFELINES
The Software Magazine

November 1982

Volume III, No. 6

Editor-in-Chief: Edward H. Currie
Editor: Jane Mellin
Circulation/Customer Service: Patricia Matthews
Design/Production: K. Gartner
Typographer: Harold Black

Communications: Bonita E. Taylor
Contributing Editor: Bill Burton
Data Processing: Lewis Tseng
Lawrence Fishman
Anne Odden
Cover by K. Gartner

DEPARTMENTS

Opinion

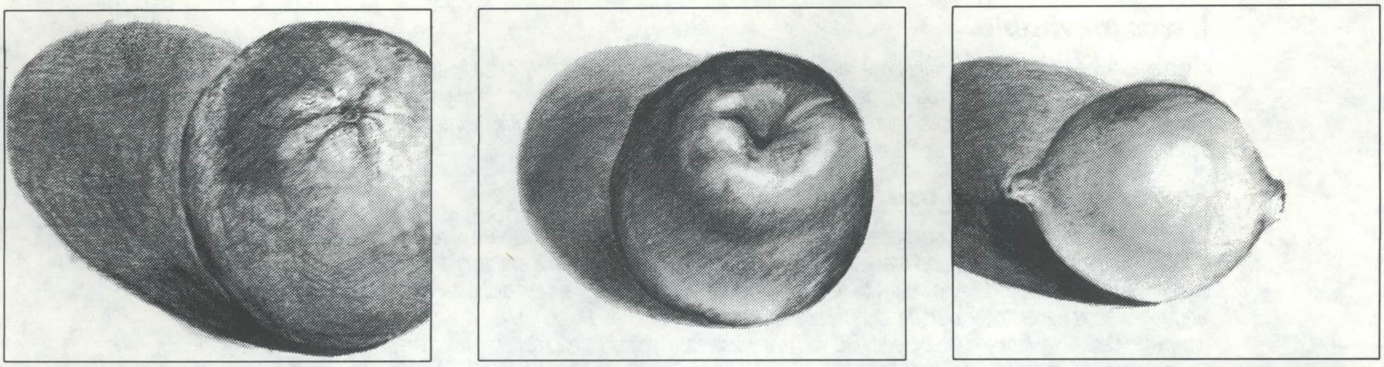
The CP/M® Users Group

Software Notes

Digital Dollars Department

Product Status Reports

Miscellaneous



FEATURES

It was inevitable.

In the beginning, there was the data base management system. Powerful, but only if you knew programming. Then came the program generator—anyone could use it, but why bother to generate poorly written BASIC programs? Now there's the best of both worlds with QUICKCODE™, the data base program generator.

Power and ease of use.

Fox & Geller's QUICKCODE™ combines power and ease of use in one neat package. It writes consise dBASE II™ programs to set up and maintain any kind of database. You can run them as is or customize them in seconds. And you'll still have all the power of dBASE II™ at your disposal: query language, report generator, and so on. But just as important: you don't need to do any programming. Just draw your data entry form on the screen and you're in business. Typical time to set up a customer list or order file: 5 minutes.

The Wordstar connection.

QUICKCODE™ also gives you the ability to transfer your dBASE II™ data into Wordstar®/Mailmerge™ files for word processing and form letters. So you can get the most from two software bestsellers: dBASE II™ and Wordstar®.

(Software dealers: DOUBLE YOUR SALES!)

That's not all . . .

There are lots of other features, like form and report generation up to 132 characters wide, four-up mailing labels, three kinds of data validation, four new data types not found in dBASE II™ itself, data base keys, and menu generators. You really have to see it to believe it.

It's your move.

Now it's up to you to take advantage of this latest development in software. Why waste any more time writing programs or paying someone to write them for you?

Fox & Geller's QUICKCODE™: $295.00.

QUICKCODE is now available for the IBM -PC with the Xedex Baby Blue Card.

QUICKCODE™

The dBASE II™ Program Generator

FOX&GELLER

Fox & Geller, Inc.
P.O. Box 1053
Teaneck, NJ 07666
201-837-0142

QUICKCODE is a trademark of Fox & Geller, Inc.
dBASE II is a trademark of Ashton-Tate.
WORDSTAR is a registered trademark of MicroPro International,
San Rafael, California USA.
MAILMERGE is a trademark of MicroPro International,
San Rafael, California USA.
IBM is a registered trademark of International Business Machines.

BASIC/Z . . .

the ultimate CP/M* compiler!

- Generates native code (8080/Z-80) for *fast* execution - 16 bit versions soon
- Sort verb is unmatched by stand-alones. 2000 elements in two seconds!
- Alpha-numeric labels, variable and function names of any length
- Chain program segments which share variables declared common
- Five data types - binary/BCD/string
- BCD floating point math - *never* a "round-off" error - precision is program definable from 6-18 digits
- Full function program editor tests syntax as you type
- Recursive, multi-line, multi-argument user defined functions
- **No royalties - No run-time charges**
- Dimension arrays dynamically (to an expression) and selectively erase
- Screen oriented editing of console input at run-time (cursor left/right/start/end, delete left/right/line, insert/change mode, and input masking available)
- Push/pop subroutine stack
- Trace and single-step debugging
- Multi-tiered error trapping even handles BDOS errors
- Cursor addressing, reverse and blinking video, erase and more are supported from source code level, with virtual hardware independence
- An extended library of over 200 "key-word" functions

For free brochure
and mini-manual:

System/z, inc.
P.O. Box 11
Richton Park, IL 60471
(312) 481-8085

* a trademark of Digital Research

System/z, inc.

Copyright © 1982, by Lifelines Publishing Corporation. No portion of this publication may be reproduced without the written permission of the publisher. The single issue price is $3.00 for copies sent to destinations in the U.S., Canada, or Mexico. The single issue price for copies sent to all other countries is $4.30. All checks should be made payable to Lifelines Publishing Corporation. Foreign checks must be in U.S. dollars, drawn on a U.S. bank; checks, money orders, VISA, and MasterCard are acceptable. All orders must be pre-paid. Please send all correspondence to the Publisher at the below address.

Lifelines (ISSN 0279-2575, USPS 597-830) is published monthly at a subscription price of $24 for twelve issues, when destined for the U.S., Canada, or Mexico, $50 when destined for any other country. Second-class postage paid at New York, New York. POSTMASTER, please send changes of address to Lifelines Publishing Corporation, 1651 Third Ave., New York, N.Y. 10028.

Lifelines - TM Lifelines Publishing Corp.
The Software Magazine - TM Lifelines Publishing Corp.
SB-80, SB-86 - TMs Lifeboat Associates.
BASIC-80, MBASIC, MS, SoftCard, COBOL-80 - TMs Microsoft, Inc.
CB80, PL/I-80, SID-86, CP/M-86, Pascal MT+, MP/M, Access Manager - TMs, CP/M and CBASIC2 registered TM - Digital Research, Inc.
DataFlex - TM Data Access Corp.
dBASE II - TM Ashton-Tate.
KIBITS - TM Bess Gerber
MailMerge, WordStar - TMs MicroPro International Corp.
PANEL - TM Roundhill Computer Systems Ltd.
PMATE, PLINK-II - TMs Phoenix Software Associates, Ltd.
Z80 - TM Zilog Corporation.
Program names are generally TMs of their authors or owners.
The CP/M Users Group is not affiliated with Digital Research, Inc.

FORTH-79

Version 2 For Z-80, CP/M (1.4 & 2.x), & NorthStar DOS Users

The complete professional software system, that meets ALL provisions of the FORTH—79 Standard (adopted Oct. 1980). Compare the many advanced features of FORTH-79 with the FORTH you are now using, or plan to buy!

FEATURES	OURS	OTHERS
79-Standard system gives source portability.	YES	
Professionally written tutorial & user manual.	200 PG.	
Screen editor with user-definable controls.	YES	
Macro-assembler with local labels.	YES	
Virtual memory.	YES	
BDOS, BIOS & console control functions (CP/M).	YES	
FORTH screen files use standard resident file format.	YES	
Double-number Standard & String extensions.	YES	
Upper/lower case keyboard input.	YES	
APPLE II/II+ version also available.	YES	
Affordable!	$99.95	
Low cost enhancement options;		
Floating-point mathematics	YES	
Tutorial reference manual		
50 functions (AM9511 compatible format)		
Hi-Res turtle-graphics (NoStar Adv. only)	YES	

FORTH-79 V.2 (requires CP/M Ver. 2.x). $99.95
ENHANCEMENT PACKAGE FOR V.2:
Floating point $ 49.95
COMBINATION PACKAGE (Base & Floating point) $139.95
(advantage users add $49.95 for Hi-Res)
(CA. res. add 6% tax; COD & dealer inquiries welcome)

MicroMotion
12077 Wilshire Blvd. # 506
L.A., CA 90025 (213) 821-4340
Specify APPLE, CP/M or Northstar
Dealer inquiries invited.

Opinion
Editorial Comments

Edward H. Currie

To Speak of Many Things ...

The response to prior editorials which also included book reviews has been excellent. Interestingly, there are now several of our authors contributing critical reports. In this month's editorial are some additional reviews which may prove of interest. In the future, however, look for a book review section in your monthly *Software Magazine, Lifelines*.

In one important subject area, perhaps the best book published to date is John Zarrella's *Language Translators - Assemblers, Compilers and Interpreters*. This book by Microcomputer Applications appears in paperback form and is one in a series called The Advanced Technology Books.

This text is unusual because John Zarrella "gives it all away" in an extremely lucid, well-designed fashion. The best description is that given by the author, in his prefactory remarks. "This book introduces language translator concepts for anyone desiring an understanding of the functions required to convert programs into machine-executable form." Concepts such as: code generation, macros, data typing, lexical analysis, syntax, parsing, semantics, optimization and symbol tables are all discussed in detail.

If John's other texts, *Operating Systems - Concepts and Principles* and *System Architecture*, are as creditable they will have a permanent place on my bookshelf.

The Brains of Men and Machines by Ernest W. Kent is a major contribution to the topic of the correlation between the way people and machines think. Kent is a professor of physiological psychology and psychopharmacology at the University of Illinois. However, the author assumes no particular background for his readers in psychology, pharmacology or physiology. The casual reader will find this material challenging but well worth the effort. This incisive treatment is a micro-tome published by McGraw-Hill as one of the Byte Books. (How's that for real medical pun-ishment ... incisive ... micro-tome ... oh well ...)

Hayes has introduced a new version of the Smartmodem which operates at either 300 or 1200 baud. This device, used with ASCOM, DMA's excellent software product, provides the ultimate in state of the art communications packages for micros.

By the way, there is an exciting new development on the communications front, relating to the transmission of software via air waves. We'll be discussing this subject in the next few months.

You'll soon be reading about a new hardware development called The Grid system. This book-sized portable microcomputer utilizes the 8088 and bubble memory with a flat screen display. Unfortunately, The Grid is not a stand alone computer. Most important, however, is the fact that the concept of a truly portable machine has taken a quantum leap forward.

The flat screen display, while impressive, is not fully equivalent to the standard CRT. Sinclair and Philips have both been experimenting with flat CRT's. Sony recently announced yet another innovative product, the Sony Watchman, which employs a similar device.

It's obvious that "flat" CRT technology will be implemented on a wide range of portable machines, offering the familiar twenty-four lines of eighty characters. Thus software designed for the traditional screen format should find its way rapidly into the realm of the portable machine.

As you read this, Digital Research will be releasing details of CP/M 3.0, to be reviewed soon. This 8080/8085/Z80 system will work best in a banked RAM environment. (A typical environment would have 96K of RAM and a maximum TPA of 62K.) CP/M 3.0 is fully compatible with CP/M 2.2 and offers time and date stamping, password protection, records lock, partial close, hashed directory access, least recently used (LRU) sector buffering, multi-sector I/O primitives, enhanced BDOS error trapping, a BDOS free space function, a BDOS program chain function, system control block, direct BIOS calls through BDOS, program and overlay loading, BIOS level I/O device assignment and resident system extensions. The IOBYTE is superseded by the assignment of I/O devices at the BIOS level. Blocking is performed in the BDOS, simplifying BIOS complexity. A help facility is also supported. Multiple CCP commands on the same line allow conditional command execution of a second command.

Networking is becoming an important area for micro users, and you should plan to add telecommunications capability to your micro in the near future. Transmission at twelve hundred baud is now within the reach of even the most modest of pocketbooks and it is joined by fine software such as ASCOM. If you haven't taken the time to access the CBBS's across the country, do so. If your own system doesn't currently support communications, go to your local computer dealer or to a friend and access a local bulletin board. The bulletin boards have begun to specialize in certain types and classes of software. You will find that the BDS C systems, for example, offer quite different features and program types than the traditional CP/M systems.

Undoubtedly there will soon be an FM receiver that plugs into a serial port on your microcomputer. This receiver will have fixed tuning to some set of local FM stations; they will broadcast programs which you can access on a subcarrier. Furthermore, by purchasing a password for a particular program, you may be able to acquire software which is not in the public domain. Data transmission mechanisms providing a

(continued on page 14)

Opinion
Talking About You

Jane V. Mellin

This past month has brought a wealth of mail-born goodies: lots of feedback from you, along with comments and suggestions for future issues. If you have something to contribute, don't hold back. Possibly others are expressing the same sentiments, and your letter could just convince us that an important need is developing. If you've got a really novel and exciting idea for helping *Lifelines/The Software Magazine* serve you, we'll implement it – and give you credit within these pages.

Calls for Help

Carl R. Camper of Colstrip, MT has a number of interesting suggestions; we were pleased at his request for a tutorial on C and reviews of some of the C's in the marketplace. It so happens that just such a series is in the works. Look for it in the coming months. (The series will cover both eight- and sixteen-bit software, ranging from some of your old favorites like Whitesmiths' C, to the exciting new Lattice C Compiler.)

Carl is having difficulty deciding which brand of disk drive can replace his old ones. Although he didn't let us know what type of drive he was searching for, other readers may be able to help him out. If you've had some good experiences with a particular brand over an extended period of time, let us know. We'll forward your input to Carl. The results will doubtless be less than perfect as an assessment of what's available, but any consistent or overwhelming trends will be reported to you.

Another request for information comes to us from Dennis P. Bowen of Skaneateles, NY, who would like to convert CPMUG NorthStar diskette formats to CP/M-80 2.2 soft sector format for his Wangwriter system. In addition, Dennis would love to find a program to allow CP/M-80 to read his Wangwriter-created text files; that way, Dennis could take advantage of the Wang full screen text editor for program entry.

A Slice of Family Life

George H. Taylor of Santa Barbara, CA is a consulting meteorologist who uses an Osborne I in his home. (He also utilizes a micro in the office.) He's been writing educational software so that his children can share his interest and enthusiasm. We can all commend George for the way he has involved his family in computing while developing some possibly remunerative applications.

There is a danger, recognized by some families I know, that the microcomputer can become another factor, like the television, in separating family members. I'm not making as radical a statement as you might at first think, and I'm not saying that microcomputing leads to divorce or juvenile delinquency or demonic possession (though some might disagree about the latter condition). I just believe we should all be conscious that microcomputing can become a very isolating activity, and make a special effort to include our loved ones in home computing activities. If you've developed some special strategies along these lines, let *Lifelines/The Software Magazine* know.

About Our Reviews

John L. Moore of Raleigh, NC begs to differ with some of the opinions James Gagne expressed in his review of Pascal/Z (see *Lifelines/The Software Magazine*). He developed a library of extensive string handling functions and procedures to augment those native to the compiler. *"Many of these functions were equivalent to those used in BASIC and all variables used in the routines were local to the individual function or procedure."* Thus, John has found Pascal/Z suitable for developing software tools. Although he agrees with our author that Pascal/Z is not perfect, he's found Ithaca Intersystems to be responsive to his problems as a user.

Several readers have complimented our reviews lately, calling them "objective", "gutsy" or "hard-hitting." Our intention, actually, is to present a product description which includes those features unique to the review's subject.

Coming Soon

We're preparing some exciting reviews for you, and we're going to touch on a few areas we really haven't explored before. Charles Strom will investigate Fancy Font, one of the new products on the market which promises to make your word processing more presentable, even pretty.

On the IBM PC front, we have numerous packages under scrutiny, including LogiQuest III, which John Howes is looking into. This relational data base management system originally ran under UCSD Pascal, and has been modified to include its own miniature operating system. Davis Foulger will present an assessment of a true hierarchical data base manager for the IBM PC, RMS.

Since we know you're eager to stay informed on the full spectrum of data base management software, we also have a number of CP/M-80 compatible packages up our reviewers' sleeves. Bob Kowitt is studying a unique product called SUPERFILE; it works backwards! If that mystifies you, look for Bob's intriguing appraisal next month. Paul Hoffman is probing Citation, advertised as a combination word processor and data manager; this product targets professional writers, and as such will receive especially intense scrutiny.

We'll soon be looking at microcomputer COBOLs. Among the more interesting projects in progress is Joseph Rothstein's evaluation of COGEN, an RM/COBOL program generator. We're also looking into a new COBOL compiler, called mdp COBOL.

Next month, we'll also be picking up our communications series with an examination of Crosstalk. ■

All About CP/M's BDOS, Part 1

Michael J. Karas

What is this "BDOS" everybody is talking about?

This series will attempt to answer that question in some detail, but first we need to understand *why* BDOS is important. Digital Research's CP/M-80 is an operating system for smaller microprocessor computers, designed to remove much of the normal computer operation drudgery experienced by the user. The operating system software embodies a "system philosophy" that structures and generalizes upon the working environment of electronics hardware. The environment presented actually allows that piece of quiet, transistorized machinery to be used at a much higher level.

The full impact of this operating system is probably felt most strongly by the typical microcomputer hacker who worked the hard way to get a computer system up and running. While the hacker was building, debugging, and integrating the pieces, the computer was just a bunch of parts interfaced together in an organized manner. However, when this composite is finally a "computer" how does it get used? The low level process of poking data into memory from a front panel or even filling, dumping, or block moving memory data with an EPROM-based "monitor program" hardly makes this computer "useful". The process of putting on disks and bringing up CP/M-80 lights the torch for computer usability. In this case the hacker experiences an elated feeling, "NOW I CAN DO SOMETHING!"

Buried in the total operating system presentation is the concept of generalization mentioned earlier. For a computer to be useful, there must be applications software to perform the jobs intended for the computer: tasks like accounting, word processing, spread sheet data analysis, or inventory control. Unfortunately, production of applications software is very, very expensive. A good package may take anywhere from one to ten years of one person's development effort to create. If the process of making an applications package had to be custom-tailored to a specific hardware environment, then there would not be affordable software for all the computers available. Generalization in the operation of a computer environment solves this problem, however. With the understanding that at a certain level "all microprocessor computer systems are alike" it is possible, with minimum constraints, to define a set of logical operations that make a computer useful.

For the Digital Research CP/M-80 operating system, this logical set of operations is defined within the BDOS portion of the operating system. Here, in about 3½K bytes of tightly written assembly language, is the "generalization converter" that takes I/O requests for hardware independent applications programs and turns them into a lower level set of simplistic hardware-oriented functions which are then processed through the BIOS. This conversion process is beneficial; it means that CP/M-80 version 2.2 can be set up to run on a typical brand XYZ computer. This conversion requires about one half of the effort needed to convert even the simplest single application package written in a hardware-dependent manner. Conclusion; software developers can make better, more sophisticated applications available at lower cost. Computer users shop in a competitive software marketplace, where there are many packages available that perform similar functions.

This presentation is intended to show the prospective applications programmer how to use most of the generalized set of "BDOS System Calls" within Digital Research's CP/M-80 version 2.2. The scheme is to describe all of the functions and use simple examples. The reader is assumed to be modestly familiar with 8080 Assembly Language Programming, as all of the examples will be given in machine language. Likewise, in this environment it is assumed by default that the prospective programmer is planning to code in assembly language. If a CP/M-80 compatible high level language is used for programming (such as Digital Research's PL/I-80 or Microsoft's BASIC-80), then of course the program interface at the "System Call" level becomes transparent to the programmer. Run time subroutines mean that the high level coded application is converted through yet another step (one major reason applications code in a high level language runs more slowly than the equivalent function written in assembly language).

Summary Of CP/M-80 System Calls

The set of system or "BDOS" I/O entry points available to the CP/M-80 programmer is complete yet simple. The primary beauty of the CP/M-80 system is this small world of completeness. Many programmers familiar with other operating systems complain that the CP/M-80 system is weak, inflexible, and incomplete. However, in a microprocessor world, the generalization level defined for the CP/M-80 system allows 85% of all microprocessor application jobs to be programmed with relative ease. Also, in my opinion, 8-bit microprocessor hardware is easily capable of performing about 90 percent of the typical tasks targeted for microcomputers. So what is this set of functions? The chart of Figure 1 summarizes, in function number order, all of the system operations specific to CP/M-80 Version 2.2 which will be covered in this presentation. In subsequent sections the functions will be grouped into categories, so that related operations reference one another.

Each function contains a certain common structure for "using" or interfacing to the CP/M-80 system. The base memory page of a CP/M-80 system memory map includes, at a specific memory address, a JUMP instruction to the CP/M-80 BDOS entry point. For most CP/M-80 systems this is address 00005H. To accomplish BDOS I/O, the number of the function is placed into the (C) register. If the parameter requires input parameters, they are passed in the (DE) register pair or the individual (E) register, depending upon whether the parameter is a word or byte value. Result information returned by some functions is sent back to the user's program in either the (A) register or the (HL) register pair, depending upon whether the value is a byte or word. The following simple program segment demonstrates the scheme used to output the 26 characters A-Z to the console screen through the use of function number 2.

```
BDOS      EQU      0005H        ;SYSTEM ENTRY
CONOUT    EQU      2            ;OUTPUT FUNCTION

          ORG      0100H        ;TPA BASE
          MVI      B,26         ;PRINT 26 COUNTER
          MVI       C,'A'       ;START WITH 'A'
;
LOOP:
          PUSH     B            ;SAVE COUNTER & LETTER
          MOV      E,C          ;LETTER TO (E) FOR OUTPUT
          MVI      C,CONOUT     ;BDOS FUNC TO (C)
          CALL     BDOS         ;GO OUTPUT
          POP      B
          INR      C            ;SEQUENCE TO NEXT CHAR
          DCR      B            ;DECREASE CHR COUNTER
          JNZ      LOOP         ;MORE TO DO IF NOT TO ZERO
          RET                   ;IMMEDIATE CCP RETURN
```

System Calls For Operator Console Input And Output

Intrinsic to the operation of any computer system, (especially CP/M-80 and its relations) is the operator console. The device provides the human interface to the machine and as such the BDOS includes a generalized set of operator communication functions to perform I/O with the console device. The various options available will each be presented with a brief example.

INPUT FROM CONSOLE KEYBOARD: Function 1.

This function waits for and reads in a character from the console device keyboard. The operator-typed character is echoed automatically back to the console display if the character is an ASCII printable character (020H to 07EH) or if it is a carriage return, line feed, back space, or tab. Note that the BDOS automatically expands tabs to columns of eight characters. Upon outputting the character for the echo, a check is made for console start/stop, CTL-S, and if so, the console input routine does not return to the user's program until another arbitrary key is depressed.

```
;CONSOLE INPUT EXAMPLE
;
CONIN     EQU      001H         ;FUNC # 1
BDOS      EQU      0005H        ;SYSTEM ENTRY

          ORG      0100H        ;START
          MVI      C,CONIN      ;FUNCTION
          CALL     BDOS         ;GO GET CHARACTER
          STA      INCHAR       ;SAVE FOR WHATEVER REASON
          RET                   ;IMMEDIATE CCP RETURN
;
INCHAR:
          DS       1            ;PLACE TO STORE INPUT
                                ;  CHAR
          END
```

OUTPUT TO CONSOLE DISPLAY: Function 2.

The ASCII character in the (E) register is sent to the console display device. The output may be any byte value, but many times the hardware driver BIOS routines automatically strip off the upper bit of the byte. Upon output the printer echo flag within BDOS is checked (CTL-P) and if it is set, the character is also sent to the printer peripheral device. Note that the BDOS automatically expands output tabs to columns of eight characters. When the character is output, a check is made for input of console start/stop, CTL-S; if this input has occurred the console output routine does not return to the user's program until another arbitrary key is depressed.

```
;CONSOLE OUTPUT EXAMPLE
;
CONOUT    EQU      002H         ;FUNC # 2
BDOS      EQU      0005H        ;SYSTEM ENTRY

          ORG      0100H        ;START
          LDA      OUTCHAR      ;GET CHARACTER TO
          MOV      E,A          ; OUTPUT
          MVI      C,CONOUT     ;FUNCTION
          CALL     BDOS         ;GO SEND CHARACTER
          RET                   ;IMMEDIATE CCP
                                ; RETURN
OUTCHAR:
          DB       'X'          ;PLACE TO GET
                                ; OUTPUT CHAR
          END
```

DIRECT USER INTERFACE TO CONSOLE: Function 6.

Some programming applications require that the BDOS not monitor the input/output character stream as we saw in functions 1 and 2. In these cases the direct I/O function is supported. The following example shows how it is used to input values and echo them until an input control-Z character is typed.

```
;DIRECT CONSOLE I/O EXAMPLE
;
DIRCIO    EQU      006H         ;FUNCTION NUMBER
BDOS      EQU      0005H        ;SYSTEM ENTRY POINT
CTLZ      EQU      'Z'-040H     ;ASCII CTL-Z CHARACTER
INPUT     EQU      0FFH         ;DIRECT INPUT FLAG

          ORG      0100H        ;CONSOLE INPUT
;
LOOP:
          MVI      E,INPUT      ;SET FOR INPUT
          MVI      C,DIRCIO     ;FUNCTION
          CALL     BDOS         ;GET INPUT OR STATUS
          ORA      A            ;IF (A)=0 NO CHAR WAS READY
          JZ       LOOP         ;CONTINUE TO WAIT FOR INPUT
          CPI      CTLZ         ;IF INPUT WAS CTL Z THEN END
          RZ                    ;CCP RETURN ON END
          MOV      E,A          ;CHARACTER TO (E) FOR OUTPUT
          MVI      C,DIRCIO     ;SAME FUNCTION NUMBER AGAIN
          CALL     BDOS         ;GO OUTPUT IT
          JMP      LOOP         ;NEXT CHARACTER INPUT LOOP
;
          END
```

PRINTING STRINGS OF CHARACTERS TO THE CONSOLE: Function 9.

Message string character sequences to be sent to the console are quite common in applications programming. Typically they are utilized for user prompt messages, program sign-on messages etc. The BDOS provides a convenient mechanism to allow the programmer to output a whole string of characters rather than having to loop with single character outputs. The string is intended to be stored in consecutive memory locations and end with the ASCII '$' character. The (DE) registers are used to point to the start of the string. The '$' signals the end of the string to display and is not sent to the console. The output bytes may be any 8-bit value, but many times the hardware driver BIOS routines automatically strip off the upper bit of the byte. Upon output of each character the printer echo flag within BDOS is checked (CTL-P) and if set the character is also sent to the printer peripheral device. Note that

(continued next page)

the BDOS automatically expands output tabs to columns of eight characters. Upon outputting each character a check is made for input of console start/stop (CTL-S) and if the check is positive, the console string output routine does not return to the user's program until another arbitrary key is depressed.

```
;CONSOLE STRING PRINT EXAMPLE
;
CONSTR   EQU      009H      ;FUNC # 9
BDOS     EQU      0005H     ;SYSTEM ENTRY
CR       EQU      0DH       ;ASCII CARRIAGE RETURN
LF       EQU      0AH       ;ASCII LINE FEED

         ORG      0100H     ;START
         LXI      D,MESSAGE ;POINT AT STRING TO SEND
         MVI      C,CONSTR  ;FUNCTION
         CALL     BDOS      ;GO SEND STRING
         RET                ;IMMEDIATE CCP RETURN
;
MESSAGE:
         DB       CR,LF,'Hello Operator',CR,LF,'$'
;
         END
```

READING A STRING OF CHARACTERS IN FROM KEYBOARD: Function 10.

The CP/M-80 console command processor (CCP), familiar to most CP/M-80 system operators, allows buffered command input with editing features. It turns out that this operation is a much-needed function for receiving strings of text from the operator console. Use of this function allows standardization of the command input functions, so the operator can easily learn the editing key functions. It also removes the pain of the applications programmer writing the same function over and over again. The read string command inputs the edited text to a buffer pointed to by the (DE) register pair. The caller specifies the maximum length desired and the BDOS returns the actual length of string entered – if carriage return is entered prior to exceeding the maximum input length. The input length is returned in both the (A) register and as part of the buffer. Bytes in the string buffer beyond the end of the entered text are uninitialized. The example shown below shows the buffer structure and how to program an input function.

The editing functions supported are the following control and/or special characters:

rub/del removes and echoes the last entered char
ctl-C initiates system reboot if first char
ctl-E echoes a CR & LF to console without putting them into buffer
ctl-H (or back space key) back spaces one char removing last entered character
ctl-J (or line feed key) terminates line input
ctl-M (or carriage return) terminates input
ctl-R retypes currently entered characters under current line
ctl-U deletes all of currently entered data and restarts buffer input on new line
ctl-X deletes all of currently entered data and restarts buffer input on same line

```
;CONSOLE INPUT BUFFER EXAMPLE
;
CONBUF   EQU      0AH       ;STRING INPUT FUNCTION
BDOS     EQU      0005H     ;SYSTEM ENTRY POINT
LENGTH   EQU      32        ;DESIRED MAXIMUM CHARACTERS

         ORG      0100H     ;START POINT
         LXI      D,STRING  ;POINT AT BUFFER AREA
         MVI      C,CONBUF  ;FUNCTION NUMBER
         CALL     BDOS      ;GO GET STRING
         RET                ;RETURN TO CCP WITHOUT
                            ;...DOING ANYTHING WITH DATA
;
;
;CONSOLE INPUT BUFFER LAYOUT
;
STRING:
         DB       LENGTH    ;MAXIMUM DESIRED INPUT LENGTH
AMOUNT:
         DS       1         ;BYTE WHERE BDOS RETURNS
                            ;..ACTUAL BYTE COUNT
STRBF:
         DS       LENGTH    ;RESERVED STORAGE FOR UP TO
                            ;"LENGTH" NUMBER OF CHARACTERS
;
         END
```

DETERMINING WHETHER THERE IS PENDING KEYBOARD INPUT: Function 11.

Some computer programs are designed to spend a lot of time processing inside of the computer or manipulating data within disk files, *without* stopping to ask the user if he or she desires to stop the processing sequence. Also it is often desirable to have a "terminate" capability for application programs, without waiting for the operator to answer a character input request. If the normal console input function is used, the user computer is not resumed until a character is already input. The console input status check function may be employed to poll the user keyboard to determine whether a character input is pending. If no input is ready, the user program is immediately resumed with an indication of whether there was a pending input. If a character is pending, a 0FFH is returned in the (A) register. Otherwise a 000H value is returned. The following example illustrates the use of console status to terminate a normally endless loop that prints the same string over and over.

```
;CONSOLE STATUS USAGE EXAMPLE
;
CONSTAT  EQU      00BH      ;FUNC # 11
CONSTR   EQU      009H      ;PRINT STRING FUNCTION
BDOS     EQU      0005H     ;SYSTEM ENTRY
CR       EQU      0DH       ;ASCII CARRIAGE RETURN
LF       EQU      0AH       ;ASCII LINE FEED

         ORG      0100H     ;START
LOOP:
         LXI      D,MESSAGE ;POINT AT STRING TO SEND
         MVI      C,CONSTR  ;FUNCTION
         CALL     BDOS      ;GO SEND STRING
         MVI      C,CONSTAT ;GET ABORT STATUS
         CALL     BDOS
         ORA      A         ;CHECK STATUS
         JZ       LOOP      ;NO KEY SO CONTINUE LOOP
         RET                ;IMMEDIATE CCP RETURN IF ABORT
;
MESSAGE:
         DB       CR,LF,'Depress any Key to STOP','$'
;
         END
```

AUXILIARY PERIPHERAL CHARACTER INPUT AND OUTPUT FUNCTIONS

The generalized CP/M-80 BDOS allows three character-by-character logical I/O devices to be attached to the computer system. This requirement stems from the fact that most computers are designed to interface to the real world through more means than just a console device. The three devices are classified as:

a) A list type device that is generally expected to be a printer of some sort. This classification is an output only device.
b) An input device supporting character input from a source other than the console. The device is specifically

an input type unit. CP/M-80 jargon refers to this device as the "READER" for no particular reason.

c) A generalized character output only device used as a specific data destination other than the console or standard list device. Some computer systems use this device, often times referred to as the "PUNCH' device as a second printer output.

The three following examples illustrate the programming techniques used to talk to each of these three devices.

```
;LIST DEVICE OUTPUT EXAMPLE
;
LIST      EQU     005H      ;FUNC # 5
BDOS      EQU     0005H     ;SYSTEM ENTRY

          ORG     0100H     ;START
          LDA     LSTCHAR   ;GET CHARACTER TO
          MOV     E,A       ;  OUTPUT
          MVI     C,LIST    ;FUNCTION
          CALL    BDOS      ;GO SEND CHARACTER
          RET               ;IMMEDIATE CCP
                            ;  RETURN
LSTCHAR:
          DB      ´L´       ;PLACE TO GET
                            ;  OUTPUT CHAR
          END
```

```
;READER DEVICE INPUT EXAMPLE
;
READER    EQU     003H      ;FUNC # 3
BDOS      EQU     0005H     ;SYSTEM ENTRY

          ORG     0100H     ;START
          MVI     C,READER  ;FUNCTION
          CALL    BDOS      ;GO GET CHARACTER
          STA     RDRCHR    ;SAVE FOR WHATEVER REASON
          RET               ;IMMEDIATE CCP RETURN
;
RDRCHR:
          DS      1         ;PLACE TO STORE INPUT CHAR
;
          END
```

```
;PUNCH DEVICE OUTPUT EXAMPLE
;
PUNCH     EQU     004H      ;FUNC # 4
BDOS      EQU     0005H     ;SYSTEM ENTRY

          ORG     0100H     ;START
          LDA     PNCHCHR   ;GET CHARACTER TO
          MOV     E,A       ;  OUTPUT
          MVI     C,PUNCH   ;FUNCTION
          CALL    BDOS      ;GO SEND CHARACTER
          RET               ;IMMEDIATE CCP
                            ;  RETURN
PNCHCHR:
          DB      ´P´       ;PLACE TO GET
                            ;  OUTPUT CHAR
          END
```

System Control BDOS Functions

This family of BDOS-supported system calls is designed to allow the programmer a degree of flexibility in manipulating the operation of general CP/M-80 environment. In general, each function here will be discussed individually because of the unique nature of each operation.

SYSTEM RESET: Function 0.

The system reset function is designed to allow restart of the CP/M-80 system command processor after a user application completes execution or is aborted. The system reset function is equivalent to a JMP to address 0000H or a CTL-C which forces a system WARM Reboot. The reboot operation de-activates all active drives except drive A: which is re-logged. Operation is extremely simple as:

```
RESET     EQU      000H       ;SYSTEM RESET FUNC
BDOS      EQU      0005H      ;SYSTEM ENTRY POINT

          ORG      0100H
          MVI      C,RESET
          JMP      BDOS        ;CALL ALSO PERMISSIBLE
                               ;EXCEPT THAT FUNCTION
                               ;DOES NOT RETURN TO USER
                               ;PROGRAM
```

GET AND SET IOBYTE: Functions 7 & 8.

The generalized CP/M-80 operating system environment communicates, via I/O, to "logical" type devices. This means that the console, list, "reader", and "punch" are just treated as generic device classifications. The CP/M-80 system allows for and supports, to a degree, the hardware containing multiple physical devices (peripherals and/or real I/O devices) within each of the generic logical device classifications. The assignment of multiple physical devices to a given classification is implemented through the IOBYTE, normally stored at address 00003H of the base page of the CP/M-80 memory. The BIOS hardware I/O software may therefore be written so that it will know which one of two printers it must address when the BDOS requires output to one of two printers. A "default standard" IOBYTE format has been adopted, based upon an 8-bit microprocessor system convention developed by Intel Corp; here it is:

		(lister)	(punch)	(reader)	(console)
Logical Devices =>		LST:	PUN:	RDR:	CON:
IOBYTE bits =>		7 6	5 4	3 2	1 0
Bit pattern 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:

The designators in the table specify the standard types of physical devices and are defined as follows:

TTY: A teletype console with keyboard, hard copy display and possibly an integral tape reader/punch
CRT: An interactive cathode ray type terminal with keyboard input and display screen
BAT: A batch processor work station with a card reader type input device and a hard copy display/output device
UC1: A user-defined alternate "console" unit
LPT: Line printer
UL1: A user-defined list device
PTR: Paper Tape Reader
UR1: User-defined "reader" character input device
UR2: User-defined "reader" character input device
PTP: Paper Tape Punch
UP1: User-defined "punch" character output device
UP2: User-defined "punch" character output device

The BDOS support for the I/O device assignment is a standard mechanism to access the IOBYTE's current value and switch it to some other value. Suppose a CP/M-80 computer had two printers connected as LST: and UL1:. If the applications program needs to switch printing output to another printer, the process could be handled as follows:

(continued next page)

```
;GET AND SET IOBYTE EXAMPLE
;
SETIOB     EQU     008H              ;SET IOBYTE FUNCTION
GETIOB     EQU     007H              ;GET IOBYTE FUNCTION
BDOS       EQU     00005H            ;SYSTEM ENTRY POINT
LSTMASK    EQU     11$00$00$00B      ;IOBYTE MASK FOR LIST
                                     ;  DEVICE
LPT        EQU     10$00$00$00B      ;BIT VALUE FOR LPT #1
UL1        EQU     11$00$00$00B      ;BIT VALUE FOR LPT #2

           ORG     0100H             ;PROGRAM START
           MVI     C,GETIOB          ;GET CURRENT IOBYTE VAL
           CALL    BDOS
           ANI     (NOT LSTMASK) AND 0FFH  ;KEEP ALL OTHER BITS
           ORI     UL1 AND LSTMASK ;SET IOBYTE FOR PRINTER #2
           MOV     E,A
           MVI     C,SETIOB          ;FUNCTION TO RESET THE IOBYTE
           CALL    BDOS
           RET                       ;IMMEDIATE CCP RETURN
;
           END
```

GET CP/M-80 VERSION NUMBER: Function 12.

Sometimes it is necessary for an applications program to "know" what version of CP/M-80 the program is running under. Versions 2.0 and above support a feature which tells the application program what the version number is. This is necessary so that version-dependent functions, such as random record file I/O, can be used if supported by the version of CP/M-80 being used. The system call to get the version number returns a two-byte value split into two parts as follows:

if (H)=0 then this is a CP/M-80 System
(H)=1 then this is an MP/M System
(L)=version number in hex
if (L)=00 then older than CP/M-80 2.0
(L)=20 then version CP/M-80 2.0
(L)=21 then version CP/M-80 2.1
(L)=22 then version CP/M-80 2.2

A program to read the CP/M-80 version number follows:

```
;VERSION NUMBER EXAMPLE
;
GETVERS    EQU     00CH      ;FUNCTION 12
BDOS       EQU     00005H    ;SYSTEM ENTRY POINT

           ORG     0100H     ;PROGRAM START
           MVI     C,GETVERS ;FETCH VERSION NO.
           CALL    BDOS
           MOV     A,L       ;SAVE CP/M-80
           STA     CURVERS   ;  VERSION NO.
           RET               ;BACK TO CCP
;
CURVERS:
           DS      1         ;STORE VERSION NO.
           END               ;  HERE
```

RESETTING THE CP/M-80 DISK SYSTEM: Function 13.

The CP/M-80 operating system contains features to control access to files on the disk drives. A directory checksum scheme, beyond the scope of this presentation, permits the operating system to determine when a disk has been changed in a drive, thus preventing the wrong disk from being written upon. However, in many cases an applications program may require disk changes as functions are changed or new files are required. This system control function permits the application to force read/write status to be set for all drives and drive A: to be logged; with this function the default disk record buffer address can be reset to its default value of 080H within the CP/M-80 base page. The following program sequence shows how to reset the disk system.

```
;RESET DISK SYSTEM EXAMPLE
;
RESET      EQU     0DH       ;FUNCTION 13
BDOS       EQU     0005H     ;SYSTEM ENTRY POINT

           ORG     0100H     ;PROGRAM START
           MVI     C,RESET   ;SET UP FUNCTION
           CALL    BDOS      ;GO RESET THE DRIVES
           RET               ;BACK TO THE CCP
;
           END
```

GET AND SET CURRENT USER CODE: Function 32.

CP/M-80 Version 2.2 permits the file system on a given drive to be partitioned into up to 15 individual directory areas, so that usage areas can be set up. For instance, the system operator could put all assembly language development programs in one user area while having disk utility programs in another. The BDOS allows the application programmer to determine the currently logged user number and to modify it if necessary. The following example sets the current user number up by one. If the highest user number is currently logged, the user 0 area is selected.

```
;GET/SET USER EXAMPLE
;
GSUSR      EQU     020H      ;FUNCTION 20
GET        EQU     0FFH      ;GET FLAG
BDOS       EQU     0005H     ;SYSTEM ENTRY POINT

           ORG     0100H     ;START UP POINT
           MVI     E,SET     ;MAKE THIS A FETCH NUM RQST
           MVI     C,GSUSR
           CALL    BDOS      ;GET CURRENT USER #
           INR     A         ;BUMP RETURNED USER UP 1
           ANI     00FH      ;MASK TO MOD(15)
           MOV     E,A       ;MOVE FOR SET TO NEW USER
           MVI     C,GSUSR
           CALL    BDOS
           RET               ;CCP GETS US BACK
;
           END
```

System Functions That Control The Disks

The data storage files for applications programs are stored on the disk drives attached to the CP/M-80 computer. The BDOS supports a number of functions that allow the state and selection status of the drives to be controlled.

SELECT DISK: Function 14.

The simplest control function selects the current disk designated as the logged or default disk. The function is equivalent to the console CCP command:

```
A>B:<cr>
B>
```

which changed the currently logged disk to drive B:. A BDOS program to for the same purpose is given in the example program of the next section below. Drive numbers correspond to the console-displayed drive designators as follows:

```
A: = Drive # 0
B: = Drive # 1

      ***

P: = Drive # 15
```

Once a drive has been selected it has its directory "activated" and is maintained in a logged in status until the next warm boot, cold boot, or disk reset BDOS function.

DETERMINE LOGGED DISK: Function 25.

An applications program can determine which disk drive is the currently logged or default drive through use of this function. The BDOS will return in the (A) register the number of the currently selected drive according to the table given above. The program segment below shows a sequence of BDOS interface code that first determines whether drive B: is selected, and if not then does a BDOS call to change it.

```
;SELECT AND POLL LOGGED DISK DRIVE EXAMPLE
;
SELECT   EQU     0EH       ;FUNCTION 14
ASKDRV   EQU     19H       ;FUNCTION 25
BDOS     EQU     0005H     ;SYSTEM ENTRY POINT

         ORG     0100H     ;PROG START
         MVI     C,ASKDRV  ;FIND OUT IF B:
         CALL    BDOS      ;  IS SELECTED
         CPI     ´B´-´A´
         RZ                ;DON´T SELECT IF
                           ;  ALREADY LOGGED
         MVI     E,´B´-´A´ ;SET TO LOG AND
         MVI     C,SELECT  ;  SELECT B:
         CALL    BDOS
         RET               ;FINISHED WITH
                           ;  ANOTHER PROG
         END
```

DRIVE STATUS SET AND RESET: Functions 28 & 37.

Drive status may be individually controlled by these functions. Operation 28 allows a the currently selected drive to be write protected (set to read/only). The process is simply:

```
WPDSK    EQU     01CH
BDOS     EQU     0005H
         MVI     C,WPDSK    ;WRITE PROTECT DISK
         CALL    BDOS
```

The write protect status of a specific disk may be removed by function 37, which deactivates the directories of each drive specified at call time. Each drive then becomes read/write again by default, but requires reactivation through reselection. The reset drive vector is a 16-bit value passed to the BDOS with a "1" bit in each bit position for a drive that requires resetting. The most significant bit of the 16 bit quantity corresponds to drive P: and the LSB to drive A:. The code sequence to reset drive B: would be:

```
RESDSK   EQU      025H
BDOS     EQU      0005H
         MVI      C,RESDSK          ;FUNCTION CODE
         LXI     D,0000$0000$0000$0010B ;DRIVE B: BITSET
         CALL     BDOS
```

GET DRIVE LOGIN AND READ ONLY VECTORS: Function 24 & 29.

The BDOS keeps track of all drives selected since the last boot or disk reset functions. These drives are considered in an online status, because the system knows immediately what the space allocation map of the drive is and whether the drive is in read/only status or not. Function 24 allows the application program to determine which subsets of the current drive complement are in this online logged status. The vector returned in the (HL) register pair is a bit map like the one above, where a "1" bit means the drive is active. The most significant bit of the 16-bit number corresponds to drive P:. The code below fetches the vector and saves it in a local data area.

```
;LOGIN VECTOR EXAMPLE
;
LOGIN    EQU     018H      ;FUNCTION 24
BDOS     EQU     0005H     ;SYSTEM ENTRY POINT

         ORG     0100H
         MVI     C,LOGIN   ;FUNCTION
         CALL    BDOS
         SHLD    LOCLOG    ;SAVE VECTOR HERE
         RET               ;TO CCP
;
LOCLOG:
         DS      2
         END
```

In a similar manner, the BDOS allows determination of which drives are in the write protected read/only status. A "1" bit in the returned vector indicates read/only status for a specific drive. The code here shows how to fetch it.

```
;READ/ONLY VECTOR EXAMPLE
;
ROVEC    EQU     01DH      ;FUNCTION 29
BDOS     EQU     0005H     ;SYSTEM ENTRY POINT

         ORG     0100H
         MVI     C,ROVEC   ;FUNCTION
         CALL    BDOS
         SHLD    LOCROV    ;SAVE VECTOR HERE
         RET               ;TO CCP
;
LOCROV:
         DS      2
         END
```

GET ALLOCATION VECTOR AND DISK PARM POINTER: Function 27 & 31.

Also provided are two more miscellaneous disk drive interface functions which permit several special types of functions to be performed. The first, function 27, returns an address in the (HL) registers that points to a bit string in memory; this bit string corresponds to the data block allocation map of the currently selected drive. The map contains one bit in each position where a block is allocated, starting with the MSB of the first byte in the string. The length of the bit string depends upon the total capacity of the drive in allocatable blocks. Function 31 permits an application to determine the characteristics of the currently selected drive. The BDOS returns an address in the (HL) registers which points to a table of 33 bytes describing the current drive. Data in the table includes such information as: number of possible directory entries on the disk, number of allocatable blocks on the disk, and, indirectly, the size of each disk block. The program below is a comprehensive example of how these functions can determine the remaining space left on a selected drive. The program stores the available space on the specified drive in the first byte of the default FCB, into memory location "KPDISK" and then exits to the CCP. The reader can adapt the code as desired.

```
;
;CP/M BDOS INTERFACE EQUATES
;
BASE     EQU     0000H      ;BASE OF CP/M SYSTEM
LOGDRIV  EQU     0004H+BASE ;LOCATION CURRENTLY LOGGED DRIVE
BDOS     EQU     0005H+BASE ;THE BDOS I/O VECTOR
SLCTDSK  EQU     14         ;SELECT DISK DRIVE
GALVEC   EQU     27         ;GET ADDRESS ALLOCATION VECTOR
GDSKP    EQU     31         ;GET ADDR OF DISK PARAMETER TABLE
;
;
         ORG     0100H
;
;
;PROGRAM TO FETCH REMAINING DISK SPACE IN KBYTES
;
```

(continued next page)

```
SPCGET:
        LDA     LOGDRIV    ;GET CURRENTLY LOGGED DRIVE, SAVE
        ANI     0FH        ;STRIP OUT USER NO.
        STA     SAVDRIV    ;SAVE CODE
;
        LDA     FCB        ;CHECK IF SAME AS S LECT
        DCR     A          ;ADJUST FCB DRVE TO MATCH SLCT DRVE
        MOV     E,A        ;..SELECT IN BDOS
        MVI     C,SLCTDSK  ;SELECT DISK FUNCTION
        CALL    BDOS
;
        MVI     C,GDSKP    ;FIND ADDR OF DISK PARAMETER HEADER
        CALL    BDOS
        LXI     B,0002H    ;INDEX TO BLOCK SHIFT FACTOR
        DAD     B
        MOV     B,M        ;(B) = BYTE BLOCK SHIFT FACTOR
        INX     H
        INX     H
        INX     H
        MOV     E,M        ;(DE) = WORD DISK BLOCK COUNT
        INX     H
        MOV     D,M
        INX     D
;
        MOV     A,B        ;ADJUST SHIFT FOR KBYTE SIZE
        SUI     03H
        LXI     H,0001H    ;CALCULATE BLOCK SIZE
SPCCAL:
        ORA     A          ;KNOW KBYTES PER BLOCK?
        JZ      SPCKNW
        DAD     H          ;DOUBLE # SECTORS PER TRACK
        DCR     A          ;DECREMENT BLOCK SHIFT
        JMP     SPCCAL
;
SPCKNW:
        MOV     C,L        ;(BC)=KBYTES PER BLOCK
        MOV     B,H
        LXI     H,0        ;INITIALIZE KPDISK
        SHLD    KPDISK
        PUSH    B          ;SAVE KBYTES/BLOCK
        PUSH    D          ;SAVE NUMBER OF BLOCKS
        MVI     C,GALVEC   ;NOW POINT TO THE ALLOCATION VECTOR
        CALL    BDOS       ;(HL)=ALLOCATION VECTOR ADDRESS
        POP     D
        POP     B
;
        SHLD    ALLSAVE    ;SAVE ALLOCATION POINTER
        MVI     H,1        ;SET MINIMUM START BIT COUNT
;
UALLOC:
        DCR     H          ;DEC BIT COUNT
        JNZ     STACT      ;STILL ACTIVE BYTE
;
        LHLD    ALLSAVE    ;GET POINTER
        MOV     A,M
        INX     H
        SHLD    ALLSAVE    ;SAVE NEW POINTER
        MVI     H,08H      ;SET BIT COUNTER TO MAX
;
STACT:
        RLC                ;GET ALLOCATION BIT TO CARRY
        JC      ALLOC      ;DON'T COUNT ALLOCATED BLOCKS
        PUSH    H
        LHLD    KPDISK     ;GET KBYTES LEFT COUNT
        DAD     B          ;ADD IN ONE MORE BLOCK COUNT
        SHLD    KPDISK
        POP     H
;
ALLOC:
        DCX     D          ;DEC TOTAL BLOCK COUNT
        MOV     L,A
        MOV     A,D
        ORA     E          ;ALL BLOCKS SCANNED YET
        MOV     A,L        ;RESTORE ALLOC BIT PATTERN
        JNZ     UALLOC     ;MORE TO COUNT
;
        LDA     SAVDRIV    ;RETURN DISK SELECT TO PREVIOUS
        MOV     E,A        ;  SELECT IN BDOS
        MVI     C,SLCTDSK  ;SELECT DISK FUNCTION
        CALL    BDOS
        RET                ;BACK TO THE CCP
;
;
;PROGRAM DATA STORAGE ALLOCATIONS
;
BLKSIZ:
        DS      2          ;STORAGE FOR ALLOCATION BLOCK SIZE
ALLSAVE:
        DS      2          ;STORAGE FOR ALLOCATION PNT SAVE
SAVDRIV:
        DS      1          ;SAVE CURRENT DISK SELECT DURING RELOG
KPDISK:
        DS      2          ;STORAGE FOR KBYTES PER DRIVE LEFT
;
        END
```

The next part in this series will present the the CP/M-80 file system as viewed from the BDOS interface aspect. The FILE CONTROL BLOCK (FCB) will be presented. In addition, preparing files for I/O and the actual I/O procedures will be presented. The series will round out to a conclusion with a comprehensive programming example, presenting a set of subroutines which permit the execution of character-by-character I/O with a file. ■

(Editorial Comments cont. from pg. 6)

channel for rapid transmission of applications programs – whether satellite, microwave, fiber optic or some combination – are inevitable and will be available in the near future.

Finally, in looking at telecommunications opportunities ask yourself about having several micros in your life, perhaps one or two at home and another at work. The two at home can back each other up and supply extra security in cases where they serve serious professional purposes, while the micro at work can be used for daily business and provide a machine to which files can be transferred when you stay at home to work.

And of course with the rapid evolution of flat screen CRT technology you will soon be able to climb on board a 747 with your portable, bubble memory, flat screen CRT 8088/Z80 machine and work comfortably and efficiently at 55,000 feet. The integral modem will permit you to dial your office upon landing and transfer files either to your machine or back to the office micro.

If you are traveling on board a 747 with public telephones, you won't even have to wait for the plane to land. Imagine getting "A>" from your New York office while you're winging your way across the Pacific at 55,000 feet to Hawaii.

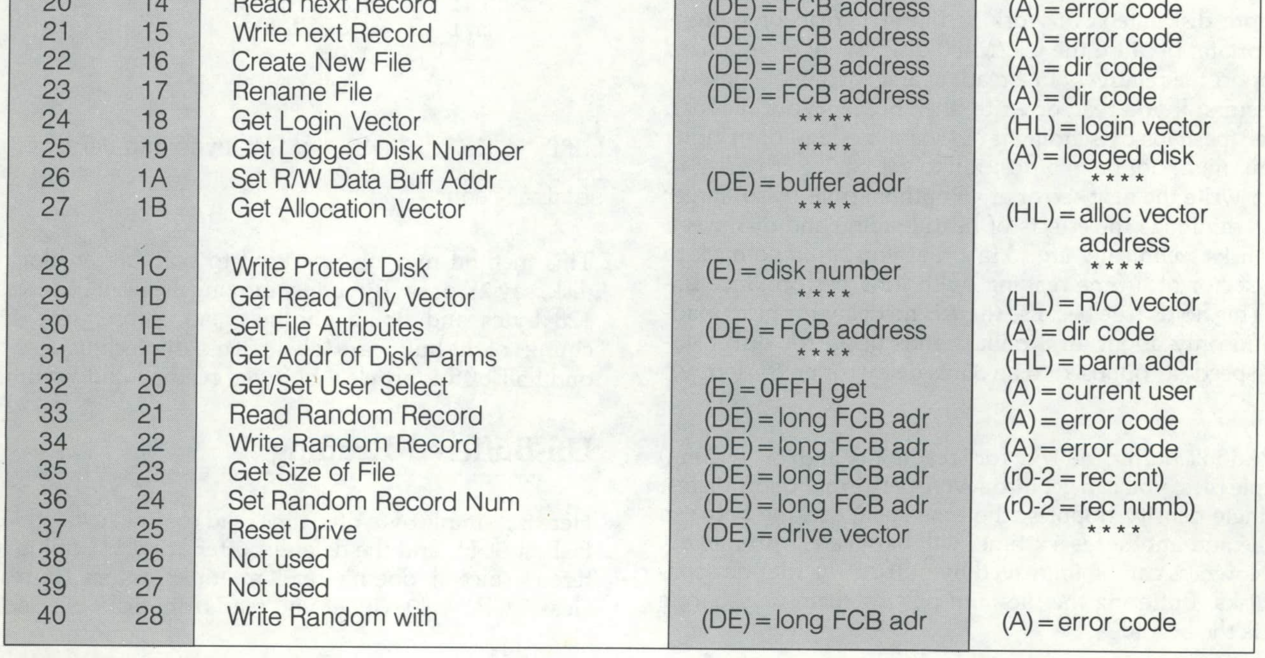
Having finished the day's work you'll fold the screen down, place it on the night stand in your hotel and head off for an evening's entertainment. The next morning your micro will awaken you with its built-in alarm clock and review your day's appointments. You'll never leave home without it … ■

Notice

The October issue was placed into the mail on September 28th. If you had any problem with the timeliness of this issue, please call our Subscription Department at (212)722-1700, or write to Lifelines/The Software Magazine Subscription Department, 1651 Third Ave., New York, N.Y. 10028. We expect to place this issue, dated November 1982, into the mail around October 27th. Each month we will print the date of the previous issue's mailing and would appreciate your help in tracking the deliveries.

FIGURE 1. DETAILED SUMMARY OF CP/M-80 2.2 SYSTEM CALLS

Function Number DEC	HEX	Function	Entry Value to BDOS Passed in (DE) or (E) regs	Return Value from BDOS Passed in (HL) or (A) register
0	00	System Reset	****	****
1	01	Console Input	****	(A) = character
2	02	Console Output	(E) = character	****
3	03	Reader Input	****	(A) = character
4	04	Punch Output	(E) = character	****
5	05	Printer Output	(E) = character	****
6	06	Direct Console I/O	(E) = 0FFH is input (E) = chr is output	(A) = character ****
7	07	Get IOBYTE	****	(A) = IOBYTE
8	08	Set IOBYTE	(E) = IOBYTE	****
9	09	Display Console String	(DE) = string addr	****
10	0A	Input Console String	(DE) = string addr	(A) = # chr input
11	0B	Get Console Status	****	(A) = 000H idle (A) = 0FFH ready
12	0C	Get CP/M Version Number	****	(HL) = Version #
13	0D	Reset Disk Subsystem	****	****
14	0E	Select Disk Drive	(E) = disk number	****
15	0F	Open a File	(DE) = FCB address	(A) = dir code
16	10	Close a File	(DE) = FCB address	(A) = dir code
17	11	Search for File	(DE) = FCB address	(A) = dir code
18	12	Search for Next	****	(A) = dir code
19	13	Delete File	(DE) = FCB address	(A) = dir code
20	14	Read next Record	(DE) = FCB address	(A) = error code
21	15	Write next Record	(DE) = FCB address	(A) = error code
22	16	Create New File	(DE) = FCB address	(A) = dir code
23	17	Rename File	(DE) = FCB address	(A) = dir code
24	18	Get Login Vector	****	(HL) = login vector
25	19	Get Logged Disk Number	****	(A) = logged disk
26	1A	Set R/W Data Buff Addr	(DE) = buffer addr	****
27	1B	Get Allocation Vector	****	(HL) = alloc vector address
28	1C	Write Protect Disk	(E) = disk number	****
29	1D	Get Read Only Vector	****	(HL) = R/O vector
30	1E	Set File Attributes	(DE) = FCB address	(A) = dir code
31	1F	Get Addr of Disk Parms	****	(HL) = parm addr
32	20	Get/Set User Select	(E) = 0FFH get	(A) = current user
33	21	Read Random Record	(DE) = long FCB adr	(A) = error code
34	22	Write Random Record	(DE) = long FCB adr	(A) = error code
35	23	Get Size of File	(DE) = long FCB adr	(r0-2 = rec cnt)
36	24	Set Random Record Num	(DE) = long FCB adr	(r0-2 = rec numb)
37	25	Reset Drive	(DE) = drive vector	****
38	26	Not used		
39	27	Not used		
40	28	Write Random with	(DE) = long FCB adr	(A) = error code



8080 Assembler Programming Tutorial, Subroutines, Part 5

Ward Christensen

More Disk I/O

This month, we'll explore some subroutines for disk I/O: read a byte at a time, and write a byte at a time. Also, I'll supply some information on using buffers other than the default disk I/O buffer at 80H for I/O.

Reading or writing a file a byte at a time can be done at several levels of complexity.

The first simply reads or writes a sector at a time, using the default buffer at 80H, and employing a pointer in memory which points to the next byte to be read or written in that buffer.

For more disk-intensive work, a buffered read or write is appropriate, because the CP/M-80 disk layout is organized to support the relatively fast reading and writing of consecutive sectors. If you read or write, then process your file for a "while" (perhaps by sending it out via a modem, or printing it), you might not be in a position to get back to the disk to read or write the next sector at an optimal time. Also, larger buffers minimize the effects of head loading and disk wear. Hard disks technically are so fast that you *could* go back to doing sector at a time reading, with little performance impact. This holds true because there is no delay for head loading, and only about an 8 milliseconds delay for disk rotational speed (as opposed to an 83ms delay for an 8" floppy).

If you don't buffer the I/O for reading (and/or writing) multiple files, you can get into severe performance problems. On single density floppies, the time spent seeking between one file and another (seek time) will be severely prolonged. This slowness can be improved by putting the files on separate disks. Buffering the files *and* putting them on separate disks is the best idea.

Another problem with multiple unbuffered files is the fact that some floppy and hard disks have sector sizes larger than 128 bytes. If you read a sector from one file, the BIOS reads perhaps 256, 512, or 1024 bytes. If you then write one sector to another file, the BIOS has to pre-read the block which that 128-byte sector fits into. If you then go back and read, the BIOS has to "flush" the modified write buffer back to disk. Only systems which employ "invisible" buffering help this situation, such as the Ithaca CACHE BIOS or the TURBODOS system, or programs running under Bob Van Valzah's track-buffering "SPEED.COM" and "FAST.COM".

Where To Read Or Write: The DMA Address

Some disk controllers directly access memory in their hardware, leaving your processor free for other work during a disk transfer. The term for this feature is "direct memory access" or DMA. While many controllers simply use the BIOS to input a character from the disk controller, and store it in memory (for example, with "mov m,a"), CP/M-80 uses the term DMA to mean simply the address at which the one 128-byte sector from a disk is read or written by the BIOS.

BDOS function 26 sets the DMA address based upon what is in the DE register. For example, a program which wants to read a sector into a buffer at address BUFF, would:

```
        lxi     d,buff
        mvi     c,setdma
        call    bdos
        .
        .
;
buff    ds      128  ;128 byte buffer
bdos    equ     5
setdma  equ     26
```

This method may also be used to read larger blocks from disk, say 2K or so. By simply moving the DMA address down 128 bytes and doing another read, subsequent 128-byte chunks of the buffer are filled. I'll use this technique in the second half of this article – buffered reading and writing.

Un-Buffered Reading

Here is a simple byte-at-a-time read routine, using the default FCB at 5CH, and the default buffer at 80H. I call it "unbuffered" since it doesn't read multiple sectors. It reads the "least" CP/M-80 will let you read from disk: one sector.

```
rdbyte  lhld    bufptr  ;get pointer
        mov     a,h     ;see if at 100h
        dcr     a       ;        a=0 if so
        jnz     noread  ;no, so get char
;
; have to read a sector
;
        lxi     d,fcb   ;point to fcb
        mvi     c,read  ;get fnc
        call    bdos    ;do the read
        ora     a       ;test for errs
        jnz     rderrs  ;       got err
        lxi     h,80h   ;re-init buf ptr
;
```

```
noread  mov     a,m      ;get a byte
        inx     h        ;point to next
        shld    bufptr   ;save pointer
        ret              ;    and return
;
; got a read error, see if eof
;
rderrs  dcr     a        ;was it 1 (=EOF)?
        jnz     rderr
;
; not an error, just eof
;
        mvi     a,´Z´-40H ;get a ctl-Z
        ret              ;return with it.
;
; got non-zero read: see if E.O.F.
;
rderr   lxi     d,ermsg  ;point to msg
        mvi     c,print  ;get print fnc
        call    bdos     ;print err msg
        jmp     0        ;exit to warm boot
;
; error msg (´$´ terminated)
;
ermsg   db       ´++ disk read error ++$´
;
; buffer pointer, init to 100H so first
;       call to RDBYTE does a sector read.
;
bufptr  dw      100h     ;init to force read
```

The only additional processing necessary is to first open the file, like this:

```
        lxi     d,fcb    ;point to FCB
        mvi     c,open   ;get fnc
        call    bdos     ;open the file
        inr     a        ;was it 0FFH?
        jz      badfile  ;     yes,
;                              no such file
```

To be a complete program, first set the code generation origin to 100H via:

```
        org     100h
```

Then load a stack pointer, such as:

```
        lxi     sp,stack
```

with stack defined somewhere (typically at the end of the program) as:

```
        ds
stack   equ
```

Recall the label is at the *end* of the stack area, since the stack works down in memory.

To make this a complete program, which for example, just lists the file to the console, add a call to RDBYTE and a call to BDOS to write the character to the console:

```
loop    call    rdbyte
        cpi      ´Z´-40h
        jz      0
        mov     e,a
        mvi     c,wrcon
        call    bdos
        jmp     loop
```

To make it a complete program, just add the error routine for not finding the file:

```
badfile lxi     d,badfmsg
        mvi     c,print
        call    bdos
        jmp     0
        db      0
badfmsg db       ´++ file not found ++$´
```

and finally, add the necessary equates which were used. I'll include all the equates that will be used anywhere in this article.

```
;memory address equates:
bdos     equ     5
fcb      equ     5CH
;console char I/O functions:
rdcon    equ     1
wrcon    equ     2
print    equ     9
;file functions:
open     equ     15
close    equ     16
erase    equ     19
read     equ     20
write    equ     21
make     equ     22
setdma   equ     26
```

Un-Buffered Writing

Un-buffered writing is very similar: it simply uses a pointer into the default buffer at 80H. The difference is in how the buffer pointer is initialized. For buffer reading, it is initialized to 100H, so the first attempt to read a byte will cause a sector read. For writing, the buffer pointer is initialized to 80H, so that we can write one sector – 128 characters – before needing to physically transfer the sector to disk.

The initialization for writing is to MAKE a file. Recall from last month that you must ERASE the file first, in case it already exists in the directory, because MAKE does not check for a duplicate entry. Thus this initialization would prepare for writing a byte at a time into a new file:

```
        lxi     d,fcb    ;point to fcb
        mvi     c,erase  ;ask bdos
        call    bdos     ;    to erase it
;
        lxi     d,fcb    ;point to fcb
        mvi     c,make   ;ask bdos to
        call    bdos     ;    make a new file.
        inr     a        ;was it 0ffh (bad?)
        jz      makerr   ;    yes, error
        .
        . the actual program itself
        .
;
; error making file
;
makerr  lxi     d,makmsg ;point to msg
        mvi     c,print  ;get print fnc
        call    bdos     ;print err msg
        jmp     0        ;exit to warm boot
;
; error msg (´$´ terminated)
;
makmsg  db       ´++ can´´t make file - ´
        db       ´directory probably full ++$´
```

(continued next page)

The call to make a file returns 0ffh if it was unable to make the file – typically because the directory is full – so that's what the error message routine tells the user.

WRBYTE simply writes a byte to the file. Here is a program which simply reads characters from the console, and directly calls WRBYTE to write the characters to disk. No correcting is allowed – if you pressed backspace, the backspace would be written to disk. End the test program by typing a control-Z, the CP/M-80 end-of-file character. The routine then ensures enough EOF characters are written so that WRBYTE will have written the sector to disk.

```
wrtest  lxi     sp,stack
        lxi     d,fcb   ;point to fcb
        mvi     c,erase ;ask bdos to
        call    bdos    ;erase file
;
        lxi     d,fcb   ;point to fcb
        mvi     c,make  ;ask bdos to
        call    bdos    ;make new file.

        inr     a       ;test 0ffh (bad?)
        jz      makerr  ;   yes, error
;
wrlp    mvi     c,rdcon ;get char from
        call    bdos    ;console
        push    psw     ;save for eof test
        call    wrbyte  ;write to disk
        pop     psw     ;get char back
        cpi     1ah     ;is it eof?
        jnz     wrlp    ;no, loop
;
; done writing characters via wrbyte, now
; pad the sector with EOF chars (1AH), so
; that wrbyte will have written the sector,
; then close the file and exit
;
wreof   lda     wbptr   ;get pointer
        cpi     80h     ;at new sector?
        jz      wrdone  ;         yes, done
        mvi     a,1ah   ;no, write another
        call    wrbyte  ;         eof char
        jmp     wreof   ;         and loop
;
; a sector containing at least one EOF char
; has been written, so just close the file
;
wrdone  lxi     d,fcb   ;point to fcb
        mvi     c,close ;get close fnc
        call    bdos    ;close it
        inr     a       ;test for 0ffH
        jz      clserr  ;if so, bad
        jmp     0       ;         else done.
;
; got close error.  Tell user.
;
clserr  lxi     d,clmsg ;point to msg
        mvi     c,print ;get print fnc
        call    bdos    ;print err msg
        jmp     0       ;exit to warm boot
;
; error msg (´$´ terminated)
;
clmsg   db       ´++ close error ++$´
```

Here is the WRBYTE subroutine itself:

```
wrbyte  lhld    bufptr  ;get pointer
        mov     m,a     ;store char
        inr     l       ;bump low pointer
        jnz     nowrite ;not at 100, no write
;
; have to write a sector
;
        lxi     d,fcb   ;point to fcb
        mvi     c,write ;get fnc
        call    bdos    ;do the write
        ora     a       ;test for errs
        jnz     wrerr   ;   got errs
        lxi     h,80h   ;re-init buf ptr
;
nowrite shld    bufptr  ;save pointer
        ret             ;   and return
;
; got a write error
;
wrerr   lxi     d,wermsg ;point to msg
        mvi     c,print ;get print fnc
        call    bdos    ;print err msg
        jmp     0       ;exit to warm boot
;
; error msg (´$´ terminated)
;
wermsg  db       ´++ disk write error ++$´
;
; error making file
;
makerr  lxi     d,makmsg ;point to msg
        mvi     c,print ;get print fnc
        call    bdos    ;print err msg
        jmp     0       ;exit to warm boot
;
; error msg (´$´ terminated)
;
makmsg  db       ´++ can´´t make file - ´
        db       ´directory probably full ++$´
;
; buffer pointer, init to 80H
;
bufptr  dw      80h     ;init for writing
```

Buffered Reading

I have a standard routine which I use for reading files using buffers larger than one sector. This is typically necessary when more than one file is concurrently open. Therefore, one (or both) of the files will use some FCB other than the default one at 5CH.

To carry out this step, I use something that large computer operating systems utilize so much – I tend to think it "glues them together" – namely control blocks.

Control blocks are areas of memory containing a specific collection of data: pointers, counters, flags, etc. The only control block used by CP/M-80 is the File Control Block. It contains: 1) fixed length strings; 2) the filename and filetype, as well as a count; 3) the file size, and several bytes or words; 4) addresses of where the file is on disk.

My control block contains three words and a byte. The layout of the control block, which I call an "EFCB" or "extended file control block", is:

word	pointer to the buffer
word	count of bytes left in buffer
byte	number of pages (256 bytes) in buffer
word	pointer to FCB with filename.

I use this in my buffered RDBYTE routine, by pointing HL to the EFCB, and calling RDBYTE.

The actual coding of an EFCB for reading would be as follows:

```
EFCB1   DW      BUFF ;buffer addr
        DW      0    ;bytes left or to write
        DB      20   ;buffer size in pp.
        DW      FCB  ;FCB addr
```

In this case, I use the system FCB. If I had another file open at the same time, it's EFCB might look like:

```
EFCB2   DW      BUFF2 ;buffer addr
        DW      0     ;bytes left or to write
        DB      20    ;buffer size in pp.
        DW      FCB2  ;FCB addr
```

The buffers themselves would have to be reserved with a size equal to that declared in the EFCB, namely 20 pages. You can let the assembler compute this size for you, instead of having to multiply 256 by 20:

```
buff    ds      256*20  ;20 sector buffer
buff2   ds      256*20  ;20 sector buffer
```

To actually read from a file:

```
        lxi     h,efcb1
        call    rdbyte
```

OR

```
        lxi     h,efcb2
        call    rdbyte
```

Here is the actual RDBYTE routine itself. To follow it, you should picture the layout of the EFCB, and, realizing that HL initially points to its first byte, keep track of where it points as RDBYTE progresses instruction by instruction. For example, the first INX H will point it to the second byte of the buffer address, etc.

Some less obvious techniques will be explained at the end of the listing.

```
;RDBYTE, HL POINTS TO EXTENDED FCB:
;
;       2 BYTE BUFFER ADDR
;       2 BYTE "BYTES LEFT" (INIT TO 0)
;
;       1 BYTE BUFFER SIZE (IN PAGES)
;       2 BYTE FCB ADDRESS
;
rdbyte  mov     e,m     ;get buffer addr
        inx     h       ;       into
        mov     d,m     ;       DE
        inx     h       ;skip buf addr
        mov     c,m     ;get bytes left
        inx     h       ;       into
        mov     b,m     ;       BC
        mov     a,b     ;get count
        ora     c       ;see if zero
        jnz     rdbnord ;no read if > 0
;
; count of bytes left = 0, so fill buffer
;
        inx     h       ;to buffer size
        mov     a,m     ;get count
        add     a       ;multiply by 2
        mov     b,a     ;sector count in b
        inx     h       ;to fcb
        push    h       ;save fcb pointer
        mov     a,m     ;get
        inx     h       ;       fcb
        mov     h,m     ;       addr
        mov     l,a     ;       to hl
;
; loop, reading sectors, until buffer
; filled or physical EOF on the file.
;
rdblp   mvi     a,1ah   ;get eof char
        stax    d       ;save in case eof
        push    b       ;save sector count
        push    d       ;save dma addr
        push    h       ;save fcb addr
        mvi     c,setdma ;set dma addr
        call    bdos    ;       into buffer
        pop     d       ;get fcb
        push    d       ;save it back
        mvi     c,read  ;request reading,
        pop     d       ;get fcb
        call    bdos    ;read one sector
        ora     a       ;test read
        pop     h       ;hl=dma, de=fcb
        pop     b       ;get sector count
        jnz     rdbret  ;got eof
        mov     a,l     ;get lo buf addr
        adi     80h     ;to next buff
        mov     l,a     ;put it back
        mov     a,h     ;add in carry
        aci     0       ;       to h, if
        mov     h,a     ;       there was one
        xchg            ;dma to de, fcb to hl
        dcr     b       ;more sectors?
        jnz     rdblp   ;yes, more
;
; the buffer is filled.  Set up the FCB
;
rdbret  pop     h       ;get fcb pointer
        dcx     h       ;to length
        mov     a,m     ;get length
        dcx     h       ;to count
        mov     m,a     ;set page count
        dcx     h       ;to lo count
        dcx     h       ;to hi fcb
        dcx     h       ;to efcb start
        jmp     rdbyte  ;loop thru again
;
; byte is in buffer, get it
;
rdbnord inx     h       ;to length
        mov     a,m     ;get length (pages)
        xchg            ;buff to hl
        add     h       ;add len to start
        mov     h,a     ;hl = end of buff
        mov     a,l     ;get end
        sub     c       ;subtract bytes left
        mov     l,a     ;put it back
        mov     a,h     ;do same for high
        sbb     b       ;pointer & count
        mov     h,a     ;hl = data pointer
        mov     a,m     ;get byte
        xchg            ;efcb back to hl
        cpi     1ah     ;eof?
        rz              ;yes, leave pointers
        dcx     b       ;decr count
        dcx     h       ;back to "bytes left"
        mov     m,b
        dcx     h
        mov     m,c     ;store back count
        ret
```

Nothing too complicated here, but a few explanations:

```
        add     a       ;multiply by 2
```

is used to take the number of 256-byte pages in the buffer, and

(continued next page)

convert this information to the number of sectors to read. "add a" simply doubles the contents of the accumulator. Note that this limits the buffer size to 127 (7FH) pages, or just under 32K. 128 or more pages would mean the "add" a would lose a bit off the top end.

The routine which gets the FCB address might bear more explanation:

```
        push    h       ;save fcb pointer
        mov     a,m     ;get
        inx     h       ;       fcb
        mov     h,m     ;       addr
        mov     l,a     ;       to hl
```

The intent is to get the FCB address into HL, to use HL as a pointer to it. To load DE from HL is easy, as was done in the first three instructions:

```
rdbyte  mov     e,m     ;get buffer addr
        inx     h       ;       into
        mov     d,m     ;       DE
```

Applying this same logic and doing this:

```
        mov     l,m     ;THIS
        inx     h       ;DOESN'T
        mov     h,m     ;WORK
```

doesn't work, because the mov l,m would clobber the current value of HL. Instead I use

```
(       mov     a,m
        inx     h
```

to get the first byte and point to the next, then

```
        mov     h,m
        mov     l,a     ;       to hl
```

which finally puts the address in HL. "mov h,m" works because HL is used as an address *before* it is clobbered by loading h.

The sector read routine keeps bumping the DMA address, so the next sector is read further into the buffer.

The routine assumes that a non-zero code on the read means end of file – it might be enhanced to check for a read error.

The only other point in need of explanation is the overall logic of using the character count: I wanted the simplicity of initializing a DW to 0, so that if I re-use an EFCB, it is as simple as zeroing out that DW. (The FCB itself would also be reinitialized, by putting a new name into it, zeroing the extent byte, etc.)

I could have used a pointer into the buffer to find out where to get the next character. But then initializing the pointer to indicate an empty buffer would have meant computing the *end* address of the buffer, and storing it. It seemed easier to use a count. Since the count is of the number of bytes *remaining*, I have to *subtract* it from the end of the buffer to point to the current character. The end of the buffer is computed by simply adding the page size of the buffer to the high register pointing to the beginning of the buffer. (You can think of a register pair as two bytes – the high byte is the "page" number, the low order, the "byte within that page". If a buffer starts at 0200H, then is three pages long, the byte past the end of the buffer is at 0200H + 0300H, or 0500H. Thus if I think of 0200H as 02 00, I can simply add 03 to 02 and get 05 00 – I can work with the high register in terms of *pages*.)

Buffered Writing

My buffered writing subroutine is similar to the buffered read routine in that it uses the same extended FCB format. But in this case the second word is a count of characters *in* the buffer, and the buffer is written whenever it is filled.

When you are done writing things a byte at a time, there will usually be bytes left in the buffer that haven't been written to disk. Thus it is necessary to flush the partial buffer to disk. For this I coded a routine called "FLUSH", also shown below.

Here is the write-byte routine. Again, it will help to have a picture of the EFCB, and keep track of what HL points to as you go from instruction to instruction.

```
wrbyte  mov     e,m     ;low buf addr to e
        inx     h       ;point to hi
        mov     d,m     ;hi buf addr to d
; de now has buffer address
        inx     h       ;to count
        mov     c,m     ;low count to c
        inx     h       ;to high count
        mov     b,m     ;hi count to b
; bc now has count of bytes in buff
        push    d       ;save fcb pointer
        xchg            ;prepare for dad
; buffer base + buffer count = next char:
        dad     b       ;to next byte
        mov     m,a     ;store it
        inx     b       ;count this char
        xchg            ;put things back
        pop     d       ;restore fcb ptr
;
; see if buffer is full, by comparing high
; byte of count to buffer size in pages,
; since the high reg represents the #
; of pages.
;
        inx     h       ;get
        mov     a,m     ;       size
        cmp     b       ;full?
        jnz     wrbnowr ;no write
;
; buffer is full.  Compute # sectors to
; write, write the buffer.
;
        add     a       ;multiply by 2
        mov     b,a     ;sector count in b
        inx     h       ;to fcb
        push    h       ;save fcb pointer
        mov     a,m     ;get
        inx     h       ;       fcb
        mov     h,m     ;       addr
        mov     l,a     ;       to hl
;
wrblp   push    d       ;save dma addr
        push    h       ;save fcb addr
        push    b
        push    h
        mvi     c,setdma ;set dma addr
        call    bdos
        pop     h
        pop     b
        pop     d       ;get fcb
        push    b
        push    d
        mvi     c,write
        call    bdos
        pop     d
        pop     b
        pop     h       ;hl=dma, de=fcb
        ora     a
        jnz     wrberr  ;got err
        mov     a,l     ;get lo addr
        adi     80h     ;to next buff
        mov     l,a     ;put back
        mov     a,h     ;get high
        aci     0       ;add carry if any
        mov     h,a     ;put high back
        xchg            ;dma to de, fcb to hl
        dcr     b       ;more sectors?
        jnz     wrblp   ;yes, more
wrbret  pop     h       ;get fcb pointer
        dcx     h       ;to length
        dcx     h       ;to count
        mvi     m,0     ;set write count
        dcx     h       ;         to 0
        mvi     m,0
        mvi     c,setdma ;reset
        lxi     d,80h   ;        dma
        call    bdos    ;        to 80h
        ret
;
wrberr  lxi     d,wrerms
        mvi     c,print
        call    bdos
        jmp     0       ;warm boot on error
;
wrerms  db      '++error in wrbyte routine++$'
;
wrbnowr dcx     h       ;to length
        mov     m,b     ;set new length

        dcx     h
        mov     m,c
        ret
;
; Call this routine to flush the buffer
; when you are done writing.
;
flush   mov     e,m
        inx     h
        mov     d,m     ;de=buf addr
        inx     h
        mov     c,m
        inx     h
        mov     b,m     ;bc=bytes in buff
        inx     h       ;to count
        mov     a,b
        ora     c
        rz              ;nothing to write
        mov     a,c     ;get low count
        add     a       ;shift "128" to carry
        mov     a,b     ;get hi count
        ral             ;x2, add in carry
        inr     a       ;allow for partial
        mov     b,a     ;b=# sectors to write
        inx     h       ;to fcb
        mov     a,m
        inx     h
        mov     h,m
        mov     l,a     ;hl=fcb
flushl  push    b
        push    d
        push    h
        mvi     c,setdma
        call    bdos
        pop     h
        pop     d
        xchg
        push    d
        push    h
        mvi     c,write
        call    bdos
        pop     h
        pop     d
        pop     b
        xchg
        ora     a
        jnz     wrberr
        push    h
        lxi     h,80h
        dad     d
        xchg
        pop     h
        dcr     b
        jnz     flushl
        xchg

        mvi     c,close
        call    bdos
        inr     a
        rnz
        mvi     c,print
        lxi     d,wrcmsg
        call    bdos
        jmp     0               ;reboot on error
;
wrcmsg  db              '++output file close error ++$'
```

The comments should pretty well explain what is going on in the write routine. Flush is mostly uncommented. You might as well get used to the "real world" of trying to figure out how an uncommented routine works.

Again, I use the "add a" trick to double the number of pages in the buffer, thus arriving at the number of sectors to write.

Also note: I reset the DMA address to 80H when I exit the write. This is because of an old CP/M-80 bug in SUBMIT that bombs if you have exited a program leaving the DMA address set at other than 80H. Actually, if you warm boot by JMPing to 0, that will reset the DMA address. However, it is nice if programs simply save the stack and return to CCP, because that saves the time of the warm boot. In *this* case it *is* necessary to reset the DMA address to 80H to allow the program to run correctly under SUBMIT (more correctly – to allow the *next* program in the SUBMIT file to execute).

In many programming examples I have loaded a word from memory pointed to by HL, or stored it back. There is no need to store it back starting at the lowest address first. For example, in WRBYTE, I am pointing *past* the EFCB byte count, so have to back up to it. Rather than:

```
wrbnowr dcx     h               ;to length
        dcx     h
        mov     m,c             ;set new length
        inx     h
        mov     m,b
        ret
```

I simply drop the updated count back into the FCB "backwards":

```
wrbnowr dcx     h               ;to length
        mov     m,b             ;set new length
        dcx     h
        mov     m,c
        ret
```

(continued next page)

The last thing is the flush routine; it simply uses the count of bytes in the buffer to compute the number of sectors to write, writes them, closes the file, and returns.

If you have tried to follow stack usage, you might have had problems. There is nothing saying that if you have PUSHed D, you have to POP D. You have to somehow account for it, for instance via POP H. I made significant use of this in the WRBYTE routine, commenting what was being pushed (e.g. DMA address, FCB address, etc.), rather than being interested in which register (B or D or H) was being PUSHed or POPped. To help check out this type of routine, I draw lines on my paper to see how the pushes and pops are paired:

```
+-------push   d        ;save dma addr
| +-----push   h        ;save fcb addr
| | +---push   b
| | | +-push   h
| | | | mvi    c,setdma ;set dma addr
| | | | call   bdos
| | | +-pop    h
| | +---pop    b
| +-----pop    d        ;get fcb
|   +---push   b
|   | +-push   d
|   | | mvi    c,write
|   | +-pop    d
|   +---pop    b
+-------pop    h        ;hl=dma, de=fcb
```

The rules for such diagramming are that the lines may be nested as much as necessary, but no lines can cross:

```
+-----push   b
| +---push   d
| | +-push   h
| | | ...
+-|-|-pop    b        ;these are
  +-|-pop    d        ;in the wrong
    +-pop    h        ;order
```

The fact that the arrows cross lets you know there is a problem. This is a very common programming bug: to pop in the same order that you pushed. The effect is that the value formerly in H will be popped into B, and what was in B will be popped into H. D will remain the same. You should only push one register and pop another if this specifically meets your needs – as it did in WRBYTE where, because all calls to BDOS have to have the value (DMA address or FCB) in DE, I pushed H with the FCB address, then popped it into DE to be ready to use it in a BDOS WRITE call.

That concludes this edition of the tutorial. If you're keeping track, I merged what had originally been outlined as section 13 – the CP/M interface – into section 12, subroutines. Thus, this is the last official piece of the tutorial.

Future tutorials will discuss program debugging using DDT (or better yet by far, SID). I'll also get into *macros*, and how they can help cut down programming time by reducing commonly needed routines to a minimum of coding. At that point, this series will end, unless extended by any questions or comments received.

KIBITS

THIS WOULD'VE BEEN ALOT SIMPLER IF MIKE HAD REMEMBERED TO TELL ME HIS CODE.
MAYBE IT'S HIS INITIALS--NO? HIS BIRTHDATE?
"INCORRECT RESPONSE" "INCORRECT RESPONSE" "INCORRECT RESP--" HUH? WHAT'S THIS?
EEEEEE
ALRIGHT, BUDDY-- HOLD IT RIGHT THERE!
BUT--
YOU'RE COMING WITH US!
...AND DONT TRY ANYTHING FUNNY!
THIS COMPUTER SECURITY IS GETTING WAY OUT OF HAND!
CS
Buss & SETON

Feature

An Alternative To CP/M-80's STAT

Thomas N. Hill

One of the most widely used programs provided with the CP/M-80 operating system has been "STAT". STAT gave us invaluable information about the free space remaining upon a disk; it told us how many precious kilobytes of disk space each file occupied. It allowed us to achieve a measure of device independence through the implementation of the IOBYTE, and (in later versions of CP/M), it provided us with some measure of control over file security.

However, in recent years an increasing number of 'extended' directory programs have been released to the public, either through The CP/M Users Group or through various Remote CP/M (RCPM) dial up systems. These extended directory programs not only provide the user with an alphabetically sorted file listing, they also provide information detailing the individual file size and the remaining free disk space. This introduction of new (and better) directory programs has reduced the role of the STAT program to the smaller tasks of IOBYTE control and the modification of file attribute flags.

Presented here are two programs which replace the functions of IOBYTE control and file attribute modification, allowing STAT to be retired to the CP/M-80 "Hall of Fame". The first program, titled "SETIO", provides a menu driven, user-friendly method for examining and modifying the IOBYTE.

The IOBYTE: What and Why

The CP/M IOBYTE, for those who are unfamiliar with it, is a byte size memory location at hexadecimal address 0003H. (In non-standard CP/M systems, the IOBYTE can be found at CPMBASE + 0003H.) The IOBYTE is divided into four fields of two bits each. Each field defines a 'logical device', which may in turn be any one of up to four physical devices. The four logical devices are termed 1) CON:, 2) RDR:, 3) PUN:, and 4) LST:. For the user's purposes, the device defined by CON: may be considered to be the one employed for primary communications with the computer. The device defined by the RDR: field is a general purpose input-only device. The PUN: field describes a device used for output only, and the LST: field controls the selection of the system printer. Note that these designations are arbitrary, particularly in the case of the RDR: and PUN:. For example, the RDR: device may actually be the output from a high speed tape drive or the output from a dedicated data logger. The primary restriction here is that the RDR: be input only and the PUN: be output only.

Each field in the IOBYTE is composed of two binary bits, capable of uniquely identifying four devices. Thus the IOBYTE provides the capability of mapping up to sixteen physical devices into four 'logical' device designators. Assuming the proper subroutines are present in the operating system to access the various physical devices, the systems programmer can thus 'mask' the physical characteristics of a peripheral from the user. The user only needs to know that if he or she wishes to get information from the data logger, for example, the IOBYTE RDR: field must merely be set to the proper value.

The four fields of the IOBYTE are defined as follows:

- Bits 0 and 1 are termed the CON: field, and, in conjunction with the proper BIOS routines, specify which physical device is considered the CP/M console.
- Bits 2 and 3 control the physical device selection for the READER device. The RDR: device is considered input only. Attempts to output to the RDR: may have interesting effects, depending upon the routines implemented by the system programmer and which program is attempting RDR: output. Note that the standard CP/M program PIP will not accept RDR: device output.
- Bits 4 and 5 control the PUN: field. The cautions described for the RDR: device also apply here, in reverse. The PUN: is considered an output only device.
- Bits 6 and 7 control the LST: device assignment. This field selects the device used for printer output from CP/M programs. In many cases a system may have two printers, one for high speed program listings and draft copies, and another for letter quality final output. The LST: field allows the user to choose between system printers.

The use of STAT to examine and alter the IOBYTE also leaves something to be desired. To make life easier for myself and for the non-technical personnel who have occasion to use my computer, I set out to create a menu-driven, user-friendly method of dealing with the IOBYTE. The program SETIO is the result. It provides four easily remembered commands (WHAT, WHERE, SET, and DEFINE) and a method for allowing the user to define his/her own device names for the various physical devices connected to the system. If the user enters the program name (SETIO) upon the CP/M command line with no following command tail, the program enters an interactive menu mode, so users unfamiliar with the program can get acquainted. If a command is placed following the SETIO program name, the program executes the command and immediately returns to the CP/M command level. (If you put the SETIO command at the CP/M command line, the program assumes you know what you are doing and won't bore you with the menu.)

Each of the SETIO commands are described below:

WHAT This command will display all possible logical to physical device assignments. The device names will be standard CP/M designations

(continued next page)

until changed by the user with the DEFINE command.

WHERE This command displays the current logical to physical device assignments. Again, the physical device names will be the standard CP/M-80 ones until changed by the user.

SET This command actually alters the IOBYTE. Invoking the SET command calls up a sub-menu displaying the four logical devices. The user is requested to select one of the four devices to change, and when the logical device has been selected, the possible assignments are displayed. The user is then queried as to the device of choice.

DEFINE This command allows the user to alter the names given to the physical devices. The program will allow the user to assign alphanumeric names of up to 24 characters to each physical device. Whenever a device has been re-defined, the program modifies its internal tables and writes itself back to the default disk.

Inside The SETIO Program

Now let's take a look at the SETIO program and how it works. Please refer to the program listing during the following discussion.

The first task is to define some useful program constants, shown in the section titled 'Program Equates'. These equates are part of a standard library file which I have developed. Following the equates section is the program proper. Each of the major program sections are outlined in the following pages.

SETIO Program entry point. This is the main program loop. After initializing various memory locations through a call to the subroutine "INIT", the program will loop through the two subroutines "COMMAND" and "EXECUTE" until the user terminates program execution.

COMMAND This is the command interpreter subroutine. A check is first made to determine if a program command was entered upon the CP/M-80 command line. If it is determined that a command was entered, the COMMAND routine converts the input line to uppercase and vectors to the lookup routine. If no command was present upon the input line, the COMMAND routine prepares an input buffer and awaits a command from the console. Upon receipt of an input line from the console, the COMMAND routine transfers control to the lookup routine.

STAT and the IOBYTE

Now that we know what the IOBYTE is, let's look at how STAT views it. STAT provides three functions relating to the IOBYTE.

1) Determine the possible field assignments.
2) Determine the current IOBYTE field assignments, and
3) Change the assignment for a particular IOBYTE field.

The possible field assignments are (using CP/M-80 naming conventions):

CON:
0 - (TTY:), console printer device,
1 - (CRT:), console assigned to the CRT device,
2 - (BAT:), batch mode: input from the RDR: and output to the CON:
3 - (UC1:), user defined console device.

RDR:
0 - (TTY:), READER is the Teletype device (usually console),
1 - (RDR:), READER is the high speed tape reader,
2 - (UR1:), user defined input device #1,
3 - (UR2:), user defined input device #2.

PUN:
0 - (TTY:), PUNCH is the Teletype device (usually console),
1 - (PUN:), PUNCH is the high speed paper punch,
2 - (UP1:), user defined output device #1,
3 - (UP2:), user defined output device #2.

LST:
0 - (TTY:), LIST output is the TTY: device (usually console),
1 - (CRT:), LIST output is sent to the CRT device,
2 - (LPT:), LIST output is sent to the Line Printer,
3 - (UL1:), user defined list device.

The STAT command to view these possible device assignments is:

```
A>STAT VAL:
```

At any one time, of the four possible devices allowed per field, only one may be assigned. To determine which one is currently assigned, the STAT command:

```
A>STAT DEV:
```

will display the current logical to physical device assignments.

To re-assign a logical device to another physical device, the general STAT command:

```
A>STAT <ldev>=<pdev>
```

is used, where <ldev> is one of the logical devices CON:, RDR:, PUN:, or LST: and <pdev> is one of the physical device names in the physical device table. Note that the colon (:) is part of the device name and must be present.

The SETIO Program And The IOBYTE

Now we finally come to the IOBYTE and the SETIO program. I don't know about you, but I find the names Digital Research assigned to their physical devices to be somewhat antique. I don't believe there are too many people who are still using Teletypes for system consoles. In many cases the CP/M device names have very little relationship to the actual physical device accessed by that particular IOBYTE field.

LOOK This is the command lookup routine. It uses the contents of the command buffer and attempts to match the input line to a command stored in the command table. The command table is formatted as the command name, in uppercase characters, followed by a byte of zero, followed by the address of the command subroutine. The last command in the table is followed by a byte of 0FFH. If no match to the input line is found, the LOOK routine returns to the MAIN program loop with an error condition.

EXECUTE This is where the actual work gets done. After the LOOK routine has found the proper command in the command table, the EXECUTE routine extracts the routine address from the table, places a return address upon the program stack, and vectors to the proper routine.

FINPROG This routine is entered when the user indicates that he or she wishes to end the program. The FINPROG routine checks to see if the user altered any device definitions. If a definition has been changed, then the modified program has to be written to the disk. The FINPROG routine first checks with the user concerning the advisability of performing this write, and if the user is in agreement, writes the program to a file named "SETIO.$$$". After writing the entire program, the old copy of SETIO.COM is erased and the new copy is renamed. If the user indicates that the modified program should not be written, the routine performs a warm boot return to CP/M.

WHERE This is the routine which performs the WHERE command. The WHERE routine uses the CP/M BDOS function call 7 to retrieve the current contents of the IOBYTE. The IOBYTE is dissected by the subroutine labeled FIELD to find the memory address of the proper device name assigned to the field and the device name string is then sent to the console. Each of the four fields is treated in a similar manner. In my system the WHERE command would produce the following output:

```
A>SETIO WHERE

Console is currently assigned to > Zenith
Z-19 CRT
Reader is currently assigned to > TTY:
Punch is currently assigned to > TTY:
List is currently assigned to > Diablo 1640
```

WHAT This routine displays to the console the possible logical to physical device assignments. The WHAT routine uses an indirect lookup table mechanism to determine the memory addresses of each of the physical device name strings. The address of each name string is then passed to the string printing routine, which displays the string to the console. On my system the WHAT command would display the following:

```
A>SETIO WHAT

CONSOLE may be assigned to the following:
1 Zenith Z-19 CRT
2 Diablo 1640
3 NULL
4 NULL
READER may be assigned to the following:
1 TTY:
2 9 Track Tape Unit
3 IMSAI Comm Link
4 NULL
PUNCH may be assigned to the following:
1 TTY:
2 9 Track Tape Unit
3 IMSAI Comm Link
4 NULL
LIST May be assigned to the following:
1 Diablo 1640
2 GE 1200 Terminet
3 Centronics 353
4 NULL
```

DEFINE The DEFINE subroutine provides the user with a method of altering the names for each of the 16 physical devices. When control passes to the DEFINE routine, a subsidiary menu is displayed:

```
Enter number of logical device:
  1. CONSOLE
  2. READER
  3. PUNCH
  4. LIST
```

The program expects an ASCII digit between 1 and 4 which it uses to index into the address table of the physical device name addresses. A section of the routine checks the input for validity and refuses to accept input other than the ASCII digits "1","2","3", or "4". When the user selects the logical device to define, the DEFINE routine will display the following (in the case of the CONSOLE):

```
CONSOLE Current assignments are:
Zenith Z-19 CRT Change to >
```

Each of the four physical devices is displayed in turn, and the user is given the option of changing the device name. If the user does not wish to change the device name, an immediate RETURN will advance to the next device without altering the current name for that device. The new device name is examined for a length greater than 24 characters and if it ex-

(continued next page)

ceeds this length, the user is requested for a new name of shorter length. After all device names have been reviewed, the DEFINE routine sets a flag indicating that the SETIO program must be re-written to disk in order to store the updated table information permanently.

SETIBYTE This is the subroutine which actually alters the IOBYTE in response to the user's commands. The SETIBYTE routine also displays the subsidiary menu of logical devices and awaits user input of an ASCII digit from 1 to 4. When the user has selected the logical device to alter, the SETIBYTE routine displays the following (again in the case of the CONSOLE):

A>SETIO SET

CONSOLE Current assignments are:
1 Zenith Z-19 CRT
2 Diablo 1640
3 NULL
4 NULL

Enter the number of the new I/O device:

The routine expects an ASCII digit from 1 to 4 in response to the entry prompt. After determining that the input is a valid digit, the SETIBYTE vectors to one of four routines (SETILST, SETICON, SETIPUN, SETIRDR); this isolates the proper IOBYTE field, clears the old setting, and places the new bit pattern in the field. Then the CP/M BDOS function 8 is used to set the new IOBYTE in memory.

INIT This is the last major subroutine. This routine initializes the various data pointers, clears or sets flags, and checks the CP/M command line buffer for a command tail. If it detects a command tail it sets a flag byte. If this flag is set, then no menu is displayed and the command tail is used as the input line to the command interpreter. After execution of the command tail, the program, instead of interactively accepting commands, will re-boot to CP/M. This allows users who are familiar with the program to bypass the menus and explanations and just perform the task at hand. (There's nothing worse than reading things you already know and have seen a dozen times before.) If the command tail flag is not set, then the INIT routine displays the selection menu and enters the interactive program loop.

Summary

Well, there it is. I have described a program designed to replace the IOBYTE handling tasks of the standard CP/M-80 program STAT with a user-friendly, menu driven routine. In another article I will describe a program designed to replace the STAT methods of altering the two file attribute flags $DIR/$SYS and $R/W/$R/O, plus the "Archive" flag implemented by MP/M II and adapted to CP/M by Kelly Smith and his "ARCHIVE" program.

I am sure that there are programmers who can see ways to improve upon the program, and I certainly welcome your input. This program has been in use both at my office and in my home system, and I think I have all the bugs worked out. If you find a major (or minor) bug that I have missed, please drop me a line.

```
TITLE ´SETIO  SETS THE CP/M IOBYTE´
0=zero O=letter O
; WRITTEN BY:   Thomas N. Hill
;               Alaska Micro Systems
;               200 Oklahoma St.
;               Anchorage, Alaska   99504
;               (907) 337-1984  (9 AM - 5 PM, AST)
; 0=zero, O=O letter
;
;Modification and Update List:
;
; 06/20/82      Version 1.0 (TNH)
;
; system equates

        CPM     EQU     0
        BDOS    EQU     CPM+5H  ; bdos entry point
        FCB1    EQU     CPM+5CH ; first File Control Block
        FCB2    EQU     CPM+6CH ; second FCB
        CBUF    EQU     CPM+80H ; command buffer
        TPA     EQU     CPM+0100H

; Non-disk I/O functions

        CONIN   EQU     1  ; console input
        CONOUT  EQU     2  ; console output
        LSTOUT  EQU     5  ; list device output
        PRTBUF  EQU     9  ; send a string to console
        RDBUF   EQU     10 ; get a string from console
        CONSTAT EQU     11 ; console status
        GETIOB  EQU     7  ; get iobyte
        SETIOB  EQU     8  ; set iobyte
        VERS    EQU     12 ; return CP/M-MP/M version #

; Disk I/O functions

        SELDSK  EQU     14 ; select disk
        OPENF   EQU     15 ; open file
        CLOSEF  EQU     16 ; close file
        DELETF  EQU     19 ; delete file
        RENAME  EQU     23 ; rename file
        READF   EQU     20 ; read record
        WRITEF  EQU     21 ; writer record
        MAKEF   EQU     22 ; create file
        SETDMA  EQU     26 ; set disk DMA address
        SIZEF   EQU     35 ; compute file size

; Those functions needing a byte argument will expect that byte
; to be in the E register.  Address arguments are passed in the
; DE register. Return codes are passed in the ACC. In general,
; a return of 0 indicates success, while a OFFH indicates failure.

; character equates

CR      EQU     0DH ; carriage return
LF      EQU     0AH ; line feed
ESC     EQU     1BH ; escape code
EOF     EQU     1AH ; end-of-file, control-z
BELL    EQU     07H ; terminal bell
BS      EQU     08H ; backspace
TAB     EQU     09H ; tab char
;
FALSE   EQU     00H
TRUE    EQU     0FFH
;

ORG     TPA

; main program loop

SETIO:  CALL    INIT            ; initialize things
MAIN:   CALL    COMMAND         ; get the input command
        JNZ     PCERR           ; print command error
        CALL    EXECUTE         ; execute the command
        LDA     CFLAG           ; command from CP/M
        ORA     A               ;    input buffer?
        JNZ     FINPROG         ; yes, return to CP/M
        JMP     MAIN            ; no, do it again.

PCERR:  LXI     D,CERMSG
        CALL    PSTRING
        LXI     D,MENU
        CALL    PSTRING
        JMP     MAIN

; subroutines begin here.

; here is the command interpreter
; it examines the contents of the buffer at 80H and if there is
; a command string from the CP/M command line, it returns to the
; main program for execution, else it requests the command from
; the user.

COMMAND:
        LDA     CFLAG           ; was a command on the input line?
        ORA     A
        JZ      COMM0           ; nope.
        LXI     H,CBUF
        LXI     D,CBUF+1        ; must absorb extra space on line
        MOV     B,A             ; convert to upper case
LP1:    INX     H
        INX     D
        LDAX    D
        CALL    UCASE
        MOV     M,A
        DCR     B
        JNZ     LP1
        LDA     CBUF            ; adjust count
        DCR     A
        STA     CBUF            ; for lost space
        JMP     COMM1

; no command on input, must get one.

COMM0:  LXI     D,PROMPT
        CALL    PSTRING
        LXI     D,IBUF          ; use internal command
        MVI     C,RDBUF         ;  buffer
        CALL    BDOS            ; get the command
        CALL    CRLF
        LXI     H,IBUF+1        ; prepare to move command string
        MOV     A,M             ; command length
        ORA     A               ; anything there?
        JZ      FINPROG         ; finish program
COMM2:  LXI     D,CBUF
        INR     A
        MOV     B,A             ; put length + 1 in B and
COMM3:  MOV     A,M             ;   use for move count
        CALL    UCASE           ; CONVERT TO UPPERCASE
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     COMM3
        XRA     A
        STAX    D               ; mark line end

; have a command, figure out what it is.

COMM1:  LXI     H,CBUF
        MOV     B,M             ; get length for use
        INX     H
        LXI     D,CTABLE        ; point to command table
LOOK:   PUSH    H               ; save command pointer
LOOK0:  LDAX    D
        ORA     A               ; zero byte from table?
        JZ      FOUND
        CPI     TRUE            ; else is end?
        JZ      COMERR          ; command error
        CMP     M
        JNZ     NEXTCOM         ; can´t be this one,
        INX     H               ;  go to next
        INX     D
        JMP     LOOK0           ; else check next char
NEXTCOM:
        INX     D
        LDAX    D
        ORA     A               ; advance to next command
        JNZ     NEXTCOM         ;   in table
        INX     D               ; first byte of command address
        INX     D               ; second byte
        INX     D               ; first of next command
        POP     H               ; re-point to command buffer start
        JMP     LOOK            ; try next command
FOUND:  POP     H               ; clean stack
        RET

COMERR: POP     H               ; clean off stack
        XRA     A
        STA     CBUF            ; set command length to zero
        STA     CFLAG
        INR     A               ; reset zero flag
        RET

; here is the EXECUTE routine.  It recovers the command address
; from the command table and branches to it, placing a return
; address on the stack first.

EXECUTE:
        INX     D               ; past end-of-command byte
        XCHG
        MOV     E,M
        INX     H
        MOV     D,M             ; command address to DE
        LXI     H,FINEXC        ; proper return address
        PUSH    H
        XCHG
        PCHL                    ; do the command
FINEXC: RET

; here is the program finish.  If no definitions were altered,
; warm boot to CP/M.  If definition(s) were altered, then save
; whole program back to disk and erase old version.

FINPROG:LDA     ALTFLAG         ; did we change a definition?
        ORA     A
        JZ      CPM

; if a definition has been changed, then we have to re-write
; the program to disk in order to save the altered names.

        LXI     D,SAVEMSG
        CALL    PSTRING         ; check a bout saving new definitions
        MVI     C,CONIN
        CALL    BDOS
        PUSH    PSW
        CALL    CRLF
        POP     PSW
        ANI     5FH             ; upper case
        CPI     'Y'
        JZ      FIN1            ; yes, save it.
        CPI     'N'
        JZ      CPM             ; nope, ignore changes
        JMP     MAIN            ; else continue with program
FIN1:   LXI     D,PFCB
        MVI     C,MAKEF
        CALL    BDOS
        INR     A
        JZ      PRERR
        LXI     D,PFCB
        MVI     C,OPENF         ; open the output file
```

```
        CALL    BDOS
        INR     A
        JZ      PRERR           ; disk error, cannot save new names
        LXI     H,SETIO
        LXI     B,(IBUF-SETIO)/128 ; sectors to save
        INX     B                  ; plus one
SAVE1:  PUSH    B
        PUSH    H
        XCHG
        MVI     C,SETDMA
        CALL    BDOS
        LXI     D,PFCB
        MVI     C,WRITEF        ; write a sector
        CALL    BDOS
        CPI     0               ; error?
        JNZ     PRERR
        POP     H
        LXI     D,80H
        DAD     D
        POP     B
        DCR     C
        JNZ     SAVE1           ; do some more
        LXI     D,PFCB
        MVI     C,CLOSEF
        CALL    BDOS            ; close it
        INR     A
        JZ      PRERR
        LXI     D,OLDFCB
        MVI     C,DELETF        ; erase old file name
        CALL    BDOS
        LXI     D,RENFCB
        MVI     C,RENAME
        CALL    BDOS            ; rename the new file
        JMP     CPM             ; finished

PRERR:  ORA     A
        JZ      PRERR1
        LXI     D,DSKERR0
        CALL    PSTRING
        JMP     CPM
PRERR1: LXI     D,DSKERR1
        CALL    PSTRING
        JMP     CPM

; here are the various command routines.

; this command displays the current IOBYTE device assignments.
; the initial program uses the standard CP/M device names, but
; the user has the option of using his own names, thru the
; "DEFINE" command.

WHERE:  MVI     C,GETIOB
        CALL    BDOS            ; current IOBYTE setting
        STA     IOBYTE          ; save it
        LXI     D,CONSOLE       ; tell about console field
        CALL    PSTRING
        LDA     IOBYTE
        MVI     B,03H           ; console field mask
        LXI     H,CNAMES        ; console names
        CALL    FIELD           ; find the proper name string
        CALL    PSTRING         ; print the name
        CALL    CRLF
        LXI     D,READER
        CALL    PSTRING
        LDA     IOBYTE
        MVI     B,0CH           ; reader field mask
        LXI     H,RNAMES        ; reader names
        CALL    FIELD
        CALL    PSTRING
        CALL    CRLF
        LXI     D,PUNCH
        CALL    PSTRING
        LDA     IOBYTE          ; same for punch
        MVI     B,30H
        LXI     H,PNAMES        ; punch names
        CALL    FIELD
        CALL    PSTRING
        CALL    CRLF
        LXI     D,LIST
        CALL    PSTRING
        LDA     IOBYTE          ; and the list field
        MVI     B,0C0H
        LXI     H,LNAMES
        CALL    FIELD
        CALL    PSTRING
        CALL    CRLF
        RET                     ; return to MAIN

; this routine displays the possible logical
; to physical device assignments.

WHAT:   LXI     D,CONMSG1
        CALL    PSTRING         ; console first
        LXI     H,CNAMES
        CALL    DEVPRNT         ; print the device list
        LXI     D,RDRMSG1
        CALL    PSTRING
        LXI     H,RNAMES
        CALL    DEVPRNT
        LXI     D,PNCMSG1
        CALL    PSTRING
        LXI     H,PNAMES
        CALL    DEVPRNT         ; ... and the punch list
        LXI     D,LSTMSG1
        CALL    PSTRING
        LXI     H,LNAMES        ; and finally the list list
DEVPRNT:LXI     B,0431H         ; B = device count, C = ASCII number
DVPT0:  PUSH    B
DVPT1:  MOV     E,M
        INX     H
        MOV     D,M             ; pick up string address
        INX     H
        PUSH    H               ; save pointer
        PUSH    D
        MOV     E,C
        MVI     C,CONOUT
        CALL    BDOS
        MVI     E,
        MVI     C,CONOUT
        CALL    BDOS
        POP     D
        CALL    PSTRING         ; print the console assignment
        CALL    CRLF
```

(continued next page)

```
        POP     H
        POP     B
        INR     C
        DCR     B               ; done yet?
        JNZ     DVPTO
        RET
; here is the DEFINE routine.  It allows the user to define
; his own names for each of the physical devices.  A limit of
; 24 chars is set on the length of the input string.
DEFINE: LXI     D,DEFWHAT
        CALL    PSTRING         ; ask which logical
        MVI     C,CONIN         ;  device to change
        CALL    BDOS            ; accept a numeric
        PUSH    PSW             ;  entry, 1-4.
        CALL    CRLF
        POP     PSW
        CALL    NUMCHK          ; check for valid digit
        JZ      DEFINE          ; not right, try again.
        CALL    DEFGET
        PUSH    D
        LXI     D,CURMSG
        CALL    PSTRING         ; "currently is:"
        POP     D
        MVI     B,4
        LXI     H,CNAMES        ; point to start of address table
        DAD     D               ; now pointing to logical
DEFG:   MOV     E,M             ;   device list
        INX     H               ; get address of current
        MOV     D,M             ;   phy. device name
        INX     H
        PUSH    B
        PUSH    H               ; save pointer
        PUSH    D               ; save string address again
        CALL    PSTRING         ; print device name
DEFG0:  LXI     D,CHANGE
        CALL    PSTRING         ; "change to->"
        LXI     D,IBUF
        MVI     C,RDBUF
        CALL    BDOS
        CALL    CRLF            ; get new name
        LDA     IBUF+1          ; check string length
        ORA     A
        JZ      DEFG4
        CPI     24              ; too big?
        JC      DEFG1           ; nope, so OK.
        LXI     D,TOOBIG
        CALL    PSTRING
        CALL    CRLF
        JMP     DEFG0
DEFG1:  LXI     H,IBUF+2        ; move string to proper place
        POP     D               ; string address
        MOV     B,A             ; bytes to move
DEFG2:  MOV     A,M
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     DEFG2           ; move the string
DEFG3:  MVI     A,´$´
        STAX    D               ; mark the end
        PUSH    D               ; must balance PUSHes
```

```
DEFG4:  POP     D               ; and POPs
        POP     H               ; recover string address table
        POP     B               ;   pointer and device count
        DCR     B
        JNZ     DEFG            ; do another one
        MVI     A,TRUE
        STA     ALTFLAG         ; tell program definitions
        RET                     ;   were altered

NUMCHK: CPI     ´1´
        JC      BADNUM
        CPI     ´5´
        JNC     BADNUM
        ORA     A               ; reset zero
        RET
BADNUM: XRA     A               ; set zero
        RET
; here is the set routine.  It will display the currently stored
; selections for the logical device selected and alter the IOBYTE
; to reflect the user´s choice.

SETIBYTE:
        LXI     D,DEFWHAT
        CALL    PSTRING
        MVI     C,CONIN
        CALL    BDOS
        PUSH    PSW
        CALL    CRLF
        POP     PSW
        CALL    NUMCHK
JZ      SETIBYTE        ; invalid answer
SETI0:  STA     LDEVNUM ; save the logical device number
        CALL    DEFGET  ; get the proper logical device printed
        LXI     H,CNAMES
        DAD     D       ; point to phys. device add. table
        PUSH    H       ; save it
        LXI     D,CURMSG
        CALL    PSTRING
        POP     H
        MVI     B,4
        CALL    DEVPRNT ; print device names
        CALL    CRLF
SETI1:  LXI     D,SELASK; ask about selection
        CALL    PSTRING
        MVI     C,CONIN
        CALL    BDOS    ; get an answer
        PUSH    PSW
        CALL    CRLF
        POP     PSW
        CALL    NUMCHK
        JZ      SETI1
        PUSH    PSW
        MVI     C,GETIOB
        CALL    BDOS    ; get current IOBYTE
        MOV     C,A
        POP     PSW
        SUI     ´0´
        DCR     A
        MOV     B,A     ; phys. device number
        LDA     LDEVNUM ; recover logical device
        SUI     ´0´
```

(cont. on page 57)

Introducing the

dBASE II® HELPLINE

The new dBASE II HELPLINE is designed to put dBASE II and QUICKCODE Users in contact with their fellow users. This new national network will enable dBASE II users to:

- ***Offer and receive advice and consultation on dBASE II programming***
- ***Exchange and standardize specific applications***
- ***Exchange information on bug reports.***
- ***Create cooperation on application developments***
- ***AND MUCH MORE!***

HERE'S HOW SIMPLY IT WORKS:

Send us your name, address, telephone, length of time using dBASE II, and type of computer configuration. We will collate, using dBASE II, of course, and make available to you a listing of other users in your region at no charge. Then it's up to you to contact them for advice, information or just to exchange ideas.

To become a member, please send the above information to:

DIRECTOR OF PRODUCT DEVELOPMENT
STANDARD SOFTWARE CORPORATION
DEPARTMENT A20
10 MAZZEO DRIVE
RANDOLPH, MA 02368

This is a free service brought to you by **Standard Software Corporation** in cooperation with four of the founding members of the Boston dBASE II Users Group (established 9/81).

David Britton   Dwayne Baye   Joseph Gannon   Glenn Meader

dBASE II IS A TRADEMARK OF ASHTON-TATE

"...Lattice C compiler is the best compiler all around that I have ever seen outside of the UNIX environment. The quality and completeness... is truly awesome."

Jason T. Linhart (*Mark of The Unicorn,* Author of Mince)

In praising Lattice C, Jason Linhart has a lot of company. It is considered by experienced users to be the definitive compiler. This 8086/8088 C Compiler supports the full C language. It is not a subset. Lattice C takes advantage of the features provided by the 16-bit 8086 instruction set and is especially suitable for applications where clear structure is crucial.

Applications of considerable complexity and power can be developed—text processing, file manipulation, data modeling, system maintenance, and much more.

Lattice C accepts source code files written in C and produces relocatable machine code in Intel's™ 8086 object module format, which can be linked together into larger programs. The Lattice C library defines a comprehensive set of I/O subroutines that implement UNIX™-compatible standard functions.

Lattice C is ideal for anyone who wants to work with or learn C—for experienced programmers who wish to enjoy the clarity and speed of C in their applications; for anyone who wants the programming capabilities of a higher level language without sacrificing program efficiency. In fact, all of the program examples listed in *The C Programming Language* by Kernighan and Ritchie can be compiled by Lattice C.

Lattice C implements the C language on all Intel 8086/8088 code-compatible microcomputers, including the IBM™ PC under DOS, MS™-DOS, and SB-86™.

For more information about Lattice C and other programs available for the IBM PC and other 8086/8088 computers, just fill out the coupon or give us a call at (212) 860-0300.

Lifeboat Associates
1651 Third Avenue
New York, New York 10028
TWX: 710-581-2524 (LBSOFT NYK)
Telex: 640693 (LBSOFT NYK)

Please send me further information on
□ Lattice C □ 16-bit software □ 8-bit software

Name Title

Company Bus. Phone

Street

City State Zip

Lifeboat Associates
World's No. 1 source of micro software

Copyright © 1982, by Lifeboat Associates. SB-86, TM Lifeboat Associates. MS, Microsoft, TM Microsoft, Inc.
UNIX, TM Bell Laboratories. Intel, TM Intel Corp. CP/M-86 TM Digital Research, Inc. IBM, TM International Business Machines, Inc.

An Introduction To Access Manager

Bruce H. Hunter

Introduction

By way of introduction, I have a small software operation in the L.A. basin where we write business programs. Our programming is dedicated to the CP/M-80 operating system. The company's latest undertaking is the creation of a data base/general accounting system. A little less than a year ago, when the program first started to develop, our initial thought was to use PL/I-80's keyed field ability to control the data base - not a bad idea if one is willing to dedicate a few years to the development of sort and search techniques. Soon after, I found myself writing my own software tools for data structures. It was an education, to say the very least!

Then I was introduced to Ashton Tate's dBASE II by Bill Hogan, a close friend, systems analyst, consultant, and CP/M expert "extraordinaire." It is very easy to get spoiled, and dBASE spoiled me for other systems. Now, the trick was to find a data base system that had some or many of the features of dBASE and would be accessible from PL/I or BASIC or any other language I was familiar with. The choice was easy. My operating systems and most of my languages were primarily Digital Research, so why not BT-80? Without a second thought, I sent off for the utility and manual. Nothing to it, right? Wrong!!

BT-80 is a binary tree retrieval system that, like any binary tree data structure system, uses pointers prolifically. My first program had close to three pages of declarations. There seemed to be more pointers in that first 180 lines than in Chapter 5 of Kernighan and Ritchie's book on "C". Now the trick of a programming utility is to make the programming task easier, and it seemed we hadn't gotten there yet.

Just as I settled down to write the world's longest segmented program on data retrieval, Digital announced Access Manager. I called Digital and asked as a favor that they ship me the package on day one of its introduction to the public. Digital didn't let me down. They shipped me Serial No. 22 in the next mail. (Thank God for credit cards!)

The first step was reading the manual. Digital's manuals. . . . well, no one has ever accused them of hiring writers. This manual is no exception, but I must admit, it is better than most. There are a lot of improvements over BT-80's manual - like an index in the back - and there are almost a couple of pictures: flowcharts. Still, like all manuals, it is written in "manualese". The only way to get into it is to put on your thigh-length manual reading boots, and wade right in.

Access Manager is a keyed file and file access package intended to operate with Digital's CB-80, Pascal/MT+, and PL/I-80 programming languages. Like BT-80, it uses the B-tree file organization, but is a lot neater. Intended for use with both CP/M and MP/M, Access Manager has both record and file locking (a must with multi-user systems). The data file organization is sequential by nature. Records are added to the end of the data file, and no reshuffling takes place except to utilize the space left by deleted records. Hogan taught me a good axiom: if you want to hurt data, move it. This form of writing avoids that problem.

Digital has really done this one right. Record lengths are whatever you want them to be, as long as they are at least four bytes long. (No more multiples of 128.) The index files *are* restricted to multiples of 128, but remember that the index files take up a lot less room than the data files, so little is lost and a great deal is to be gained in potential standardization. Now, you can only have 124 key (index) values per index file. Personally, I try to keep them down to two or three. The maximum index file size is 8 megabytes. I hope these "restrictions" won't discourage you!

I'm going to restrict most of my commentary to single user systems, so please bear in mind that the MP/M II restrictions are much larger than the constraints imposed by CP/M-80. Maximum data file length is also 8 megabytes. Open files are 20 data and 10 index maximum. These constraints seem large by ancient (last year's) standards, but with $1300 Winchesters and extended addressing nearly taken for granted, they are more than realistic.

Access Manager owes its origins to B-tree structures, which came into use during in the late 60's. Developed more or less simultaneously at Sperry Univac, Case Western Reserve University, Control Data, Stanford, and Boeing Scientific Research Labs, the meaning of the name is lost in obscurity. It has been claimed that the 'B' stands for balanced, broad, bushy and even Boeing. The B-tree structure was quick to gain acceptance, and became the heart of many mainframe data systems, including IBM's highly touted VSAM system. Going even further back into computer prehistory, its roots stem most certainly from the binary tree. (*Editor's Note:* Donald Knuth in *The Art of Computer Programming,* Volume 3, ascribes the "B" to Bayer, who first described the structure in 1972.)

Binary Trees

The binary tree utilizes a data structure called a node. The node consists of the information to be stored (which may or may not be a file key), and a right and left pointer. The pointers "point" to the next lower level pair of nodes, the smaller value to the left, and the greater value to the right. There is an esoteric vocabulary associated with binary trees and the relative levels of the nodes. Concentrating on one node (any node), it would be known as the "father" relative to the two nodes to which it is pointing on the next level down. Those two nodes to which the "father" is pointing are known as the "sons." (The exclusivity of the male family members here I can only ascribe to blatant male chauvinism.)

Looking at that one symbolic node again, all nodes preceding that one in upper levels are known as "ancestors," while all those below that node (except for the sons) are "descendants." The topmost node is called the "root" node. Terminal nodes (without sons) are known as leaves.

Without any housekeeping algorithms, binary trees soon become unbalanced, one side becoming longer or going into more levels than the other. Now, the fewer number of accesses it takes to retrieve the data you want, the more efficient the tree. In an unbalanced tree, efficiency is soon lost, so intricate housekeeping routines are required to keep the tree in order and balanced. To determine the maximum number of accesses it will take to retrieve a piece of information, the height of the tree is the key. The height of a balanced tree is related to the log of the number of nodes in the tree:

$$\text{Height} < = \log d\ ((N + 1)/2)$$

For example, the height of a tree of a half million values is 18; that is to say, it would take no more than 18 accesses to retrieve a value. When the tree is to be searched for a value, the topmost node (or root) is examined and compared with the search value. Whether the examined node is greater than or less than the search value will determine the next search path. "Less than" will cause the search to go to the left pointer, and "more than" will branch to the right one, all the way down the tree, until either the value wanted is found or until a null pointer is encountered. If a null pointer is encountered, the search is terminated. You have to try to write a binary tree to appreciate the complexity of the task. If you think a simple goto will get you into trouble, you have never pointed a pointer! Pointers point at direct memory locations. I don't want to discourage you, but you can blow up your system in just a line or two of code!

B-Trees

As you will recall, the binary tree node consists of information (which may or may not be a file key) and two pointers. The B-tree has the distinction of using a node with multiple keys. While the binary tree node has one key and two pointers, the B-tree has one more pointer than it has keys. Therefore, a node of order d will have 2d keys and 2d + 1 pointers. (Because of the multiple number of keys, groupings of keys are referred to as "pages.") The number of keys in each node (or page) determines the efficiency of the search. The more keys per page, the fewer levels to complete the tree. Page size is a function of key length and the number of sectors allocated to each page. A great deal of housekeeping is required by the system to keep the tree balanced. The writer of a B-tree system must balance the time spent in housekeeping relative to the time saved in the search.

BT-80 and B+Tree

Back in 1980 or so, Digital brought out its BT-80, a B-Tree record retrieval system. Aimed at PL/I-80 programmers, it utilized the B+Tree system. The B+Tree is yet another level of sophistication in B-Tree organization. It organizes the root node and all levels of nodes, short of the lowest level, to page the keys located at the bottom of their structure. The upper nodes or sequence nodes split the ranges of nodes below them. It would be analogous to the first page of a telephone directory telling you which page held the location of the page which listed the name (key) that you were looking for. Sounds a bit complex at first, and it is! But remember, the trick is to cut down on the number of tries to complete the look up, not to make the understanding of the system simpler. This organization very substantially reduces the height of the tree structure. Using "d" as half the number of keys per page and "n" as the number of keys, the height of a tree is more or less:

$$\text{Tree height} < = 1 + \log d\ ((N + 1)/2)$$

(The more or less is dependent on the fullness of the nodes, another subject altogether. But before getting sidetracked on this one and maybe even B*-Trees we'll get back to B+ Trees.)

Using the above formula, a half million keys could be searched in no more than four accesses. That is fast!! Remember the binary tree, with its 18 accesses per half million pieces of information! To add to the wonder of it all, the sequence nodes are maintained in left and right ascending order. The final search for the key is done sequentially. Compare that to the binary tree, which must be traversed in "preorder," a tricky algorithm of descending the tree left-most at each level and then right across that level before a left descent again. The B+Tree presumably can be returned in order by a simple left to right sequential read.

"Son Of BT-80"

I talked to the people at Digital about their new offspring. As I suspected, Access Manager is indeed "son of BT-80." They are justifiably proud of their newest accomplishment. Combining B-tree hierarchy and indexed sequential access method (ISAM), they have brought the sophistication of the VSAM file system to the micro. My friend, Hogan, refers to PL/I as a "behemoth." It is probably the largest language in the history of computer languages (although ADA may surpass it in time). PL/I-80 is subset "G" of PL/I, and I find it a wondrous irony that PL/I-80 does not support ISAM or VSAM files like the full set; but with Access Manager used in conjunction, PL/I-80 now has the power of its mainframe brother.

Program Notes

A look at the listing at the end of the article will give a fair idea of what is involved in programming with AM-80. The data structure at the top of the code serves both as a classic PL/I structure and as a data buffer. A pointer is prepared to point to the buffer later in the code. The precompiler command, %include am80extr.pli, inserts the declarations of the AM-80 functions and parameters, thoughtfully saving the programmer a quarter page of typing. After the declarations, the parameters of the system are defined. The number of buffers is set at three, the system minimum. Two index files will be used, having 4 sectors of 1024 bytes each. Digital recommends standardizing this number to make AM-80 programs accessible by all programmers using AM-80. I heartily join them in this recommendation. One data file will be used, and error trapping through the program (rather than directly) has been chosen. No file or record locking will be used in this single user program, and time out is not necessary – since it is used only in multi-user environments.

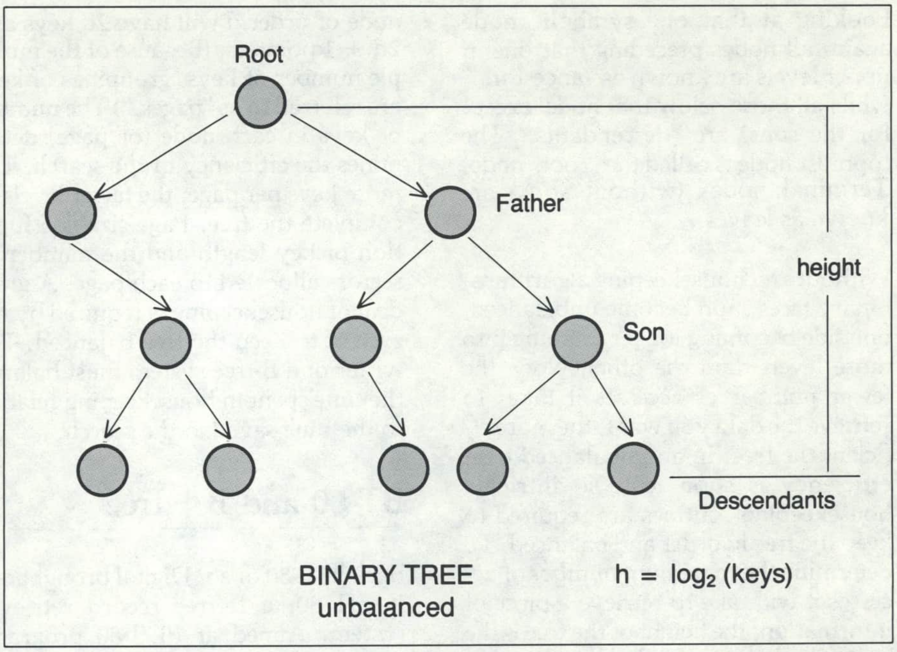
The system is initialized with the "in-

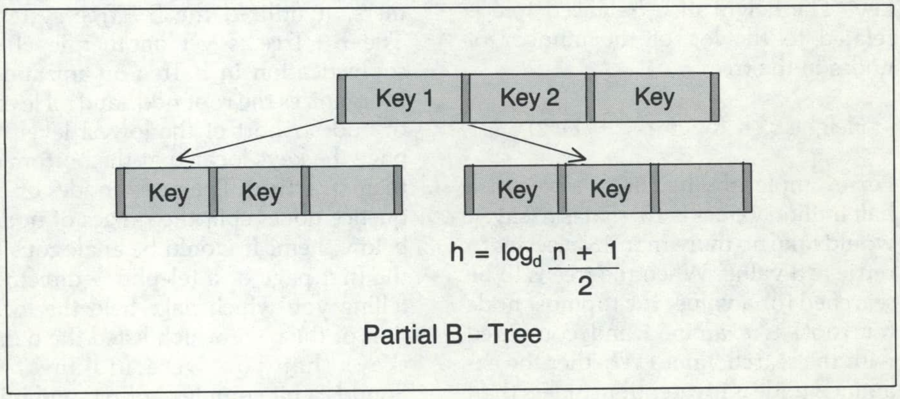
(continued next page)

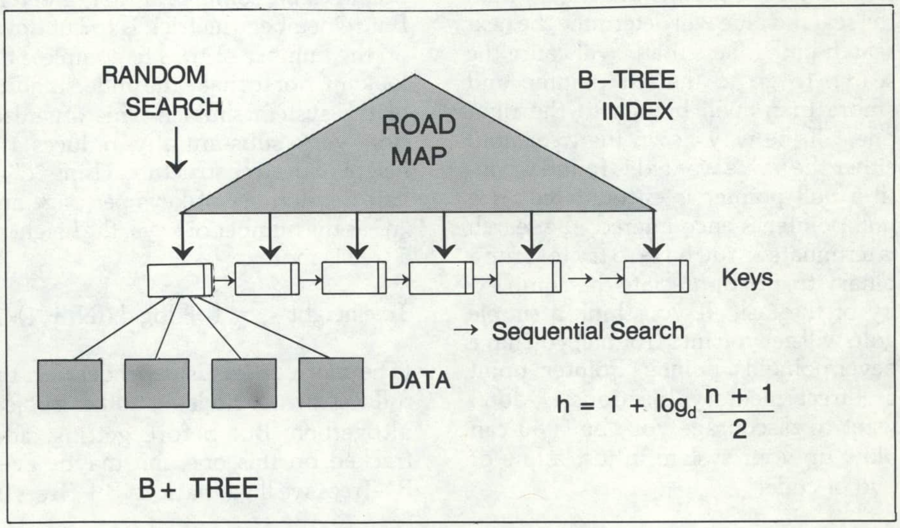
tusr" and "setup" routines. Intusr assigns the program identification, another goodie ignored by the single user system. Setup does the actual initialization as well as setting up the buffer area. To get even with the multi-user parameters that don't get used, setup is ignored by multi-user systems. The parameters passed to intusr are the program i.d., error request and time out function. In this single user program only the error trapping request is significant, but all must be used. Setup needs the numbers of buffers, index files, index sectors, and data files.

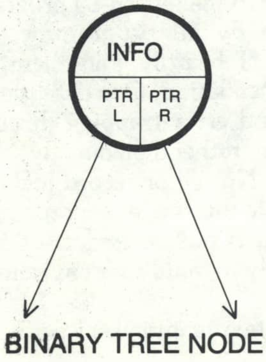
The data file(s) is opened with "opendat." Opendat, as the name implies, opens the data file and assigns a data file number if you request it. There is another function, "oprdat," which will open a blown file and rebuild it. Handy for people who have a compulsion to cold boot in the middle of a disk write. Parameters passed to the routine are the file number, the lock request, file name, and record length. The index files are opened with the "opnidx" routine. Parameters are file number request (-1), the index file name, the maximum key length, the key type, alpha or numeric, and the choice of a duplicate key suffix. If chosen, the duplicate key suffix will allow the use of up to 65535 duplicate keys. If not chosen, a duplicate key will simply not be written to the index file.

Writing the data and indices is nearly as simple. The next available record number is required and it is returned by the function "newrec" which takes as parameters the file number and lock request. Newrec, along with the file number and data buffer pointer, is all that's necessary required to write the data to disk using "wrtdat." To put the key records to disk requires the "addkey" routine: addkey writes the new key value and its record number to the index file.









```
/**************************************************************************/
/********                   CHTINIT1.PLI                     ************/
/********               CHART INITILISATION                  ************/
/********                first attempt at                    ************/
/********                 Access Manager                     ************/
/********                  Sept 6, 1982                      ************/
/********      Bruce H Hunter/ HUNTER STRUCTURED SOFTWARE    ************/
/********                Copyright(C) 1982                   ************/
/********                      <<*>>                         ************/
/********    A trivial program to initilise Chart of Accounts ***********/
/**************************************************************************/
/***                                                                   ***/
/***     linkage note: link chtinit1,am80pl/I.IRL[S,A],AM80BUF.IRL     ***/
/***                                                                   ***/
/**************************************************************************/
```

```
chart:
    proc options (main);
    dcl
        data_buf_ptr    pointer,
        1       data_buffer,
            3   act_name            char(32) var,
            3   act_no              char (4),
            3   address,
                5       street      char(32) var,
                5       city        char(24) var,
                5       state       char(2),
                5       zip         char(5),
            3   principal           char(24);
                                    /* -----
                                       123  byte total */

    %replace
        MAX_KEY_LEN by 48,
        NAME_LEN by 14,
        TRUE by   ´1´b,
        FALSE by ´0´B,
        CLEAR by ´^1´;
    %include ´am80extr.pli´;
    dcl
        no_lock fixed,
        file_name char(NAME_LEN) var,
        (name_index, nbr_index, name_key) char(128) var,
        (n_buf, n_keys, n_sec, n_data_files, erropt, progid, time_out) fixed,
        (drn, file_no, recd_len, act_name_len, nbr_len, name_type) fixed,
        (nbr_type, name_dup, nbr_dup, act_name_key, act_nbr_key) fixed;

    n_buf = 3;  /* index file buffers (name & number) */
    n_keys = 2; /* index files */
    n_sec = 4;  /* 1024 byte index file record length */
    n_data_files = 1;
    erropt = 1; /* trap user errors */
    progid = -1;/* program assigned id number request, ignored
                        in a single user system */
    time_out = 0;/* background server delay also ignored */
    no_lock = 0;  /* file lock request (no file lock) */

    /**************************************************************
    initilize system
    */
    progid = intusr(progid, erropt, time_out); /* progid not used in
                        single user system */
    if errcod() ~= 0 then
        call error(1);
    if setup (n_buf, n_keys, n_sec, n_data_files) ~= 0 then
        call error(2);

    menu:
        begin;
        dcl
            choice  fixed;
        put list (CLEAR);
        put skip edit(
            ´           MENU´,
            ´           ****´,
            ´´,
            ´1          open data file and key files´,
            ´´,
            ´2          open files and update´,
            ´´,
            ´3          close files & return to menu´)
            (8 (skip,col(24),a));
        retry:
        put skip (2) list (´^iinput choice´);
        get list (choice);
        if choice < 1 ¦ choice > 3 then
            goto retry;
        goto q(choice);           /* case */
        q(1):
        call open();
        goto menu;
        q(2):
        call open();
        call update();
        goto menu;
        q(3):
        call close();
        end menu;
    open:
        proc;
        dcl
            reply char (1);

        put list (CLEAR);
        put skip(4) edit(
            ´           Open Index and Data File´,
            ´           ************************´)
            (2 (skip, col(20), a));
            /**/
        file_no = -1;   /* auto file no assignment */
        recd_len = 128;
        file_name = ´chartdb.dat´;
        /* open data file (create or update) */
```

Its parameters are the index file number or key number, data file number, data lock request, the key value, and the data record number. Associated functions are "delkey" to delete keys and data record numbers from index file records and updptr, to change the record number associated with the key.

To flush the data buffers and write the directory information to disk, the closing procedure is called. "Clsdat" closes the data file using only the file number. Clsdat closes the data file and updates to disk while "savdat" updates the disk without closing the file. The index files are closed by "clsidx", passing it only the index file; an additional function, "savidx" saves the information to disk without closing the file number.

Most routines pass an error code. I found the error codes straightforward and easily understood. "Errcod" returns AM-80 error codes. There are about 70 error codes or messages, most of which are quite specific. Nothing as nebulous as "conversion," my old enemy in PL/I.

There are numerous additional routines such as "nokeys" and "nmnods" which return the number of keys in an index file and the number of nodes respectively. There has been a great deal of work put into file and record locking for multi-user applications. Locking and unlocking functions are supplied, and a number of lock requests exist. Lock codes can be set to no lock as the listing has done or shared record or file locks which allow any number of users to share data. Exclusive locks can also be set. The use of a shared lock prevents a single user from using an exclusive lock.

Digital has supplied a wealth of functions for index key searching. Getkey looks for an exact match and returns the corresponding data record number. What if we don't have an exact target key? Serkey will look for the first key which is equal to or greater than the target key. Neat! Give it a target key of "0" or "A" and it should start an ascending sort. Befkey finds the key that precedes the target key and aftkey the following. Frstkey and laskey return the first and last keys. Prevkey goes to the key before the key just accessed; nxtkey, of course, goes to the one after.

(continued next page)

```
  file_no = opndat (file_no, no_lock, file_name, recd_len); /* note:
                  the file_no is optional in single user systems */
  if errcod() ~= 0 then
        call error(3);
  data_buf_ptr = addr(data_buffer); /* set pointer to buffer */
  /*
  open index files
  */
  name_index = 'name.idx';
  nbr_index = 'nbr.idx';
  act_name_len = 11;
  nbr_len = 4;
  name_type = 0; /*alphanumeric key */
  nbr_type = 0;
  name_dup = 1;  /*add duplicate key suffex if necessary */
  nbr_dup = 0;   /*no duplicate acount number suffex */
  act_name_key = opnidx(-1,name_index,act_name_len,name_type,name_dup);
  act_nbr_key  = opnidx(-1, nbr_index, nbr_len, nbr_type, nbr_dup);
  end open;
update:
  proc;
  dcl
      key_name char(9) var,
      (reply, ud_code, ud_code2) fixed;

  put list (CLEAR);
  put skip(4) edit (
                  'Update',
      '           ######',
      '',
      'enter EOF to quit')
      (4(skip, col(24), a));
  do while (TRUE);
     oops:
      put skip list ('account name :');
      get edit (act_name) (a);
      if act_name = 'EOF' | act_name = 'eof' then
         goto menu;
      put skip list ('account number :');
      get list (act_no);
      put skip list ('street :');
      get edit (street) (a);
      put skip list ('city :');
      get edit (city) (a);
      put skip list ('state :');
      get list (state);
      put skip list ('zip :');
      get list (zip);
     put skip list ('principal or contact :');
     get edit (principal) (a);
     /**/
     put skip (3) list ('^i^i*Verification*');
     put skip (2) list (act_name, '^i', act_no);
     put skip list (street, ' ',city,',  ',state,'  ',zip);
     put skip list (principal);
     put skip (2) list (' enter 1 for corrections :');
     get list (reply);
     if reply = 1 then
        goto oops;
     /* write data to mr. disk */
     drn = newrec (file_no, no_lock); /*
        returns the next availabvle data record number */
     if errcod() ~= 0 then
        call error(3);
     if wrtdat(file_no, drn, data_buf_ptr) ~= 0 then
        call error(4);
     name_key = substr(act_name,1,9);
     /* add key values to key files */
     ud_code = addkey(act_name_key,file_no,no_lock,act_name_key,drn)
     if ud_code = 2 then
        put skip list('index value ',act_name,' all ready in file')
     ud_code2 = addkey(act_nbr_key,file_no,no_lock,act_nbr_key,drn);
     if ud_code2 = 2 then
        put skip list('index value ',act_no,' all ready in file');
     end;/*dowhile*/
 end update;
 close:
     proc;

     if clsdat(file_no) ~= 0 then
        call error(6);
     if clsidx(act_name_key) ~= 0 then
        call error(7);
     if clsidx(act_nbr_key) ~= 0 then
        call error (8);
     stop;
     end close;
 error:
     proc(location);
     dcl
        location fixed;
     put skip(3) edit
       ('^iError ', errcod(), ' at code location ', location)
          (a, f(4), a, f(3));
     stop;
     end error;
 end chart;
 /*that's all folks*/
```

Digital Research has done its homework well. The AM-80 package is broad and well thought out. It has subroutine libraries, interfaces, and systems processes for PL/I, CB80, and Pascal/MT+. A shared multiple-user background server is supplied as well as a number of multiple-user resident system processes. Single user programs will normally only require linking the appropriate AM-80 library of subroutines and the buffer area. A recreate utility has also been supplied to recreate blown, or as DRI likes to call them, 'corrupted', data and index files.

In my opinion, the Access Manager is the finest accessible data storage and retrieval system available today for microcomputers. It is easy to implement and does the majority of housekeeping itself, not leaving this chore to the programmer. Its speed and capacity should go unrivaled for some time. We will see if I am still smiling next month after bringing up the search routines.

References

BT-80 Record Retrieval System Reference Manual
Digital Research, Inc.

Access Manager Programmer's Guide
Same as above

Data Structures and PL/I Programming
Augenstine & Tenenbaum

Fundamentals of Data Structures
Horowitz & Sahni

"The Ubiquitous B-Tree"
Comer ■

Renew

We're looking forward to hearing from any of you December subscribers who haven't called or written. If your subscription started with the December '81 issue you should have received a letter and reader survey from us, urging you to renew. You can see that Lifelines/The Software Magazine has given you value this past year and we're expecting your support again. Make your holidays merrier and send your check right away. Or get out your VISA or MasterCard and call Lifelines/The Software Magazine Subscription Dept. at (212) 722-1700. The address is: 1651 Third Ave., New York, N.Y. 10028.

MicroMoneymaker's Forum

Digital Dollars Department

Charles E. Sherman

Good Ideas In Early Returns From Readers
and
Entrepreneur Of The Month: Edward S. Greenberg
and
A Guaranteed Money Saving Tip With Abject Apology From Yours Truly

The Early Returns

At last! The first information sheets are rolling in, chock-full of thought-provoking information for us all. In fact, this month's profiled entrepreneur was discovered in the first batch. His one-line description of what he does was just enough to pique our interest, and an interview turned up pay-dirt. To tell the truth, even the interview was his idea. His info sheet came with a letter making suggestions for the column, one of which was that we interview our own readers. This seemed like a good idea, and it only seemed fair that he be the guinea-pig. Bingo!

Circumstances cause an obtrusive lag between the time you post some hot tid-bit to us and the time you will read about it in these pages. For this we can thank our turtle-express mail multiplied by three: from you to *Lifelines/The Software Magazine*, then to me, then back to *Lifelines*. This is compounded by the 30 days between each issue. So when you send us something wonderful, please be patient; it will take two or maybe three months before we can recycle it. For comparison purposes, Byte Magazine takes about six months for turnaround. Some wonderful day *Lifelines* will discover the Source or some other electronic mailbox, and we can all begin to live up to our potential for speedy communications.

The statistical summaries from the first responses are suggestive, but not yet reliable. Our numbers are still quite low, which I am confident is due to the above mentioned time warp and not to the fact that you are all sitting on your hands.

And now, may we have the envelope, please:

1. The early returns of the MicroMoneyMaker's Information Sheets indicate that our responding readers have been involved with microcomputers during a period of one to seven years, with an average of 3.4 years.

2. Half of the people who responded are consultants of some kind. It is fairly typical for a respondent to enjoy ancillary income, in many cases derived from the individual's own programming, via sales and/or royalties, while a few supplement with hardware sales or business support services. The respondents who aren't consultants are mostly either salaried people who hope and expect to become independent consultants imminently, or who provide services in support of other businesses or professions. Those services usually consist of routine business needs, such as accounting, mailing, or filing.

3. Nearly 100% of the respondents rely upon word processing software (mostly WordStar). Just over 62% use a spreadsheet program (mostly SuperCalc), and about the same percentage use a data-base (mostly dBASE II). I'm glad to hear that so many of you rely upon word processing, because I just happen to be scrutinizing all fifteen of the major CP/M-80 programs now on the market, including a customized version of WordStar, and may have some interesting tales to tell in future columns.

Many readers had some ideas for micromoneymaking, but the descriptions were frequently far too vague or too abbreviated. This month's entries in the imagination sweepstakes are:

1. Maintain a political information database, especially focusing on campaign contributions. The main idea is to generate data about politician's pals, to see who is influencing whom.

2. Become a broker for programming services, communicating between customers and clients via telecommunications.

3. Set up journal indexing, abstracting, and reprint systems for research professionals. I gather a tailored data base is intended, if I understand this one-liner. This same person has DataStar and says professional indexers are still looking for the ideal indexing program. I wonder if they've checked out Documate/Plus? If you're still out there, let us know.

4. Customize, install, and train people to use the off-the-shelf software that most business users can't make functional for their needs and to their complete satisfaction.

5. Computerized city map and routing service. Subscribers could call in deliveries for the day and receive back the most efficient routing.

(continued next page)

As I said, the descriptions of what you do, as well as your suggestions for what could be done, seem to suffer from a certain verbal stinginess. However, something is better than nothing, and even these fragments may stimulate a hot idea somewhere. I hope more of you will write to enter the imagination sweepstakes, or to tell us what you do for money. When you do write, *please* flesh those ideas out with a bit more detail.

Here's an example of what I mean. One part-time entrepreneur says "I index books and journals." Now that's tantalizing, but leaves us with nothing but unanswered questions. Why don't authors index their own books? Now I have heard that a writer is a fool if he tries to index his own book, and that it takes objectivity coupled with a special knack or special training. Is this true? Why? What is it like to work as an indexer? Does it pay well? Is there much demand for good indexers?

We would rather you wrote in with fragmentary information than with none at all, but we would all prefer something to get our teeth into, okay?

Profile of the Month

Edward S. Greenberg has more business than he can handle, but it keeps expanding anyway. Eddy calls himself a consultant, but thinks of himself more as a teacher. However you pronounce it, he's up to his ears in action. Operating in New York city under the name *Distributed Data Processing*, he teaches insurance salesmen how to use micros and off-the-shelf software in simple applications aimed mostly at sales, with a few tricks thrown in for administration. He says success comes easily because he keeps things simple and strictly on a practical level. He believes this is what people want and need, and their enthusiastic response seems to bear him out. His business has grown rapidly by word of mouth alone – he has never advertised.

In the insurance field, salesmen depend upon "illustrations" which consist of a list of figures tailored for each potential customer's facts. These figures will show the customer the difference between payments made over 20 years under term insurance, and payments made over a like period under whole-life insurance, with some variables thrown in to cover various tax approaches. In the past, salesmen have had to call on the potential customer, get the facts, send off to the head office, and wait a week or two for the figures to come back. With a micro and Ed's own little BASIC program, they can get the figures immediately for some instant show-and-sell. Obviously, this is a lot more effective. As an irresistible bonus, the salesmen can also use their micros to keep track of customer lists, contacts, write letters, and so on.

Typically, Eddy will show an insurance sales office the advantages of having a micro around. Then he offers to set up the equipment and teach their people how to use it, and they fall all over themselves signing up. Most contracts start with a $2500 retainer, for which he studies the office and its needs, then makes recommendations. Then he sells them the equipment and teaches them how to use it. All they have to do is sit back, learn, and pay. And pay. He'll go in once a week for a while, then taper off, eventually switching over to service by phone. After showing people how to use the micro and run his basic program to generate sales "illustrations," Eddy will teach them how to use Visicalc and WordStar. If they are eager for still more, they can go into dBASE II (mostly for customer lists and the like). Eddy's main research activity is staying a jump ahead of the customers, and thinking up new applications to help them.

Not bad for a 24 year-old high school drop-out, is it? That's what I like about the microcomputer world: it is a frothing, wide-open field in which formal education and credentials mean a lot less than creative ingenuity and enterprise. For the future, Eddy is working on ways to make himself available to more clients by setting up a database which his clients can subscribe to.

Eddy Greenberg believes that there is room for hundreds like himself in the insurance industry alone, as there are over 50,000 salesmen in the country and the degree of micro penetration is less than one half of one percent. Then there are all the other industries which could benefit from his approach: off the top, he suggests the garment industry or real estate. He also thinks there is a big need for business appraisals and valuations. I suggest pension plan valuations, which are *required* as part of every divorce in California (150,000 annually) and most other states. Actually, almost any field will do, because the potential is totally untapped. Take any field you used to be in, or any field you are interested in, or any field you have contacts in, and start teaching them some simple micro applications that will help their process; preferably to increase sales.

Eddy thinks the secret to success lies in keeping things simple and practical. He says, "The most important advice I can give prospective consultants is to come away from sophisticated programming and intricate applications, and go in for education. The big need is at the bottom, at the beginning, where the mass of the population is right now. Show them how to use micros to make simple applications work for them."

Guaranteed Money-Saving Tip After Abject Apology

Well, I made a mistake and I admit it. I'm sorry. I owe you this apology and I hope you'll give me another chance.

In the last two columns I've been oiling around with words of artifice, scheming and conniving to figure out some cleverly conducive way to get you to send me information about what you do and what you know, and presumptuously prying even into your imaginations. Who am I, Internal Revenue or something? Some hustler looking for a scam-sketch? A computerized busybody? How can you write things to me if you don't know to whom you're writing or why?

Unlike another contributor to this magazine (whose name I will not mention but you can spell it with alphanumeric bits and pieces from two old Z80s), I am not an anonymous person. What I am in fact, as far as you are concerned, is a sheep in wolf's clothing.

I am not and never have been a computer whiz, nor even a hacker (though some of my more ignorant and hysterical acquaintances might disagree). I aman entrepreneur who makes it with his micro. What I mean is that I make money and books. I made my last four books on my micro *including* the galleys for pasteup. I look at a word processor not as a super-typewriter, but as a cross between a chord organ and a junior typesetting machine. Ten years and several books later – law yourself, hydroponics, health – I stand before you as an ex-legal tired and re-retired entrepreneur who is playing and plying on the silicon shores for the same reasons most of you are: it's more fun and it's what's happening.

One of two current books in progress for a noted micro publisher: *Elegant Computer Print or How to Make a Good Impression With Your Microcomputer With Professional Quality Brochures, Flyers, Catalogues, Reports, Newsletters, Proposals, Letters, or Anything You Want People to Pay Attention To*. I may have to whittle the title down a bit. The main message for *Lifelines'* entrepreneurial readers is that everything you send out in writing should look good. You want to impress people with the fact that you are *professional* and competent. It is especially inappropriate for any microcomputer professional to send out written material that is less than excellent, because you are your own best example of the power and potential of the microcomputer. You have to practice what you preach.

With a little encouragement from you, we could go into this in more detail in a future column.

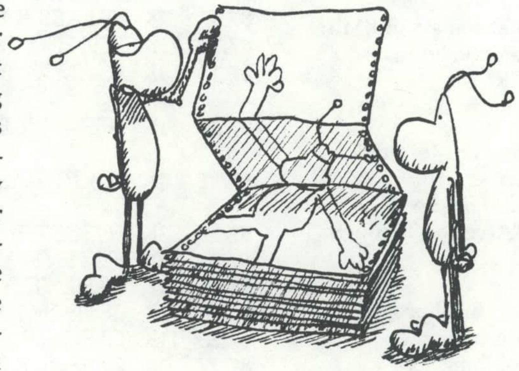
My column in *Lifelines* is an outgrowth of a notion that one of the most interesting things about micros may be the variety of practical, entrepreneurial uses to which they are put by active and talented enthusiasts. That's you folks. What you are doing is interesting to me, and probably will be interesting to your fellow *Lifelines* readers. That's why I keep wheedling for information from you about what you do, what you have heard, what you can imagine.

I am proceeding on the theory stated in the first column (last August) that *Lifelines* is read by talented people who are, or are about to be, or should be making their way microwise. I assume that you or your clients are involved with entrepreneurial micromanship. I can write about things that I think will be useful and important to you, but I would rather write about things that you tell me are interesting. Please send in your suggestions.

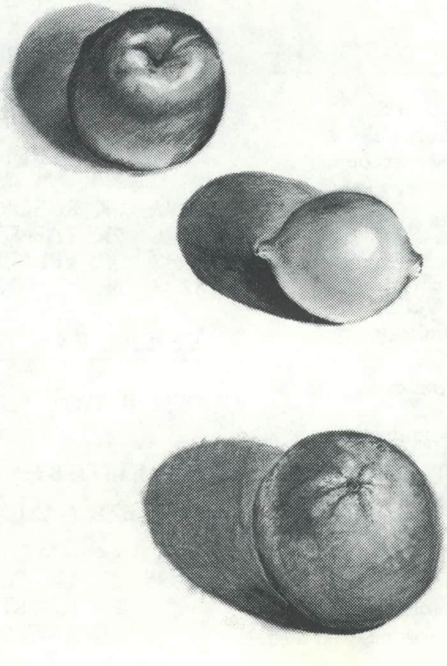
Now you know more about me, how about telling me about you?

As promised, here at last is this month's guaranteed money-saving tip. When you write to us, use the pre-paid envelope that *Lifelines/The Software Magazine* generously includes with every magazine. The more often you write, the more money you will save.

See you next month. Bye. ■



“I FOUND THAT BUG IN MY PROGRAM...”



Make your systems sales soar!

STOK PILOT™

A language for creators.

STOK PILOT can make your final product more saleable and keep it sold.

This unique langauge allows straightforward creation of a friendly interface between novice users and CP/M for any program application. Using STOK PILOT, a software designer can easily produce a hand-holding menu-driven front-end supervisor with built in tutorials to help guide the novice end-user through a complex application. The actual end-user does not have to know STOK PILOT.

STOK PILOT opens the door for companies to evolve structured approaches to meeting their own training needs.

STOK PILOT can simulate any application program for the purpose of tutoring the user. This allows training and guidance without the necessity of working on and possibly damaging live data. Then, when the user is ready, STOK PILOT can call the actual application program.

STOK PILOT is a superset of the PILOT CAI language. STOK PILOT was designed with the syntax and structure of PILOT because PILOT is an easy to use and easy to learn language.

STOK PILOT can be thought of as an interactive SUBMIT facility. One program written in STOK PILOT can control an entire session without ever entering cumbersome CP/M commands. The commands can be dynamically formed at run time by STOK PILOT.

Menus can be created that will allow the user to select various programs. STOK PILOT can chain to any program or utility written in any language, leaving the entire TPA available, and regain control when the other program ends.

STOK PILOT can check for the existence of a file before it actually calls the application that needs it. Let STOK PILOT see to it that the proper disks have been inserted by the user.

The package includes a well written and indexed 75 page manual. All the instructions are explained in full detail. Many useful programming examples are also included.

Complete 8 inch CP/M format disk and manual retails for $129.95. The manual is available alone for $14.95 and is deductible from a future order. NY residents please add sales tax.

Toll free order line: (800) 431-1953 ext 183
In NY (800) 942-1935 ext 183

C.O.D. - Mastercard - Visa

SSI
Stok Software Inc.
17 West 17th Street
New York, NY 10011
(212) 243-1444

Dealer inquiries invited
CP/M is TM of Digital Research

Volumes 86-90, Catalogues and Abstracts

CP/M Users Group

Volume 86

DESCRIPTION: BUSINESSMASTER II, Volume 1 of 5:
documentation.

SUBMITTED BY: Bud Aaron
BUSINESSMASTER
1207 Elm Ave, Suite M
Carlsbad, CA 92008

NO.	SIZE	NAME	COMMENTS
	2K	-CATALOG.086	CONTENTS OF VOL. 086
	3K	ABSTRACT.086	Release to public domain of BMII.
	5K	U-G-FORM.LIB	CPMUG Submission Form.
	2K	CRCKLIST.086	File of all file CRC's.
	2K	CRCK.COM	to check files on this disk.
86.1	2K	APPENDA.DOC	Diskette handling tips for novices.
86.2	3K	APPENDB.DOC	Disk File Capacities.
86.3	4K	APPENDC.DOC	Data File Diskette Set-up.
86.4	2K	APPENDD.DOC	Programs Not Accessed Through The Menu.
86.5	10K	APPENDE.DOC	Compilation of BUSINESSMASTER II progs.
86.6	18K	APPENDF.DOC	General File Information.
86.7	14K	APPENDG.DOC	Forms for establishing acct. #'s etc.
86.8	2K	APPENDH.DOC	How to increase size of a data file.
86.9	7K	GLOSSARY.DOC	Glossary of terms
86.10	12K	HIERARCH.DOC	What program calls what, and what CPMUG disk are they on?
86.11	132K	NEW.DOC	Comprehensive overall system doc.

Note: All .DOC files are meant to be printed with WordStar; however they are pre-formatted containing only ↑J page ejects, and a few WordStar"dot" commands, such as .pa and .pl. **Also note**: the SUB files mentioned in APPENDE.DOC were not included with the submitted material. I made a "COMPILE.SUB" for each volume containing .BAS programs.

Volume 87

DESCRIPTION: BUSINESSMASTER II, Volume 2 of 5:
initialization, startup, modification and maintenance; inventory/fixed asset accounts; mailing list;

SUBMITTED BY: See above

No.	SIZE	NAME	COMMENTS
	4K	-CATALOG.087	CONTENTS OF VOL. 87
	5K	U-G-FORM.LIB	Users Group Submission Form.
	2K	CRCKLIST.087	File of all file CRC's.
	2K	CRCK.COM	to check files on this disk.
	2K	COMPILE.SUB	To compile all programs.
87.1	1K	ALL.BAS	Include FORMAT and CONTROL
87.2	4K	BIZMII.BAS	Master Menu BUSINESSMASTER II. (bizmii)
87.3	2K	CHECK.BAS	Check for needed files
87.4	1K	CONTROL.BAS	Set up control chars for term.
87.5	1K	FORMAT.BAS	Set common and formats.
87.6	1K	READFILE.BAS	Read name, date, formats and tabs.
87.7	6K	CGDEPCAL.BAS	Depreciation Calculator
87.8	11K	CGENTRY.BAS	Capital Goods (Fixed Asset) Entry Program
87.9	3K	CGFM4562.BAS	IRS Form 4652 - Depreciation Printer
87.10	4K	CGRPT.BAS	Depreciation Report Printer
87.11	3K	CGSORT.BAS	Capital Goods Entry Sort Program
87.12	3K	CRLABELS.BAS	Cust. Mailing Label Printing Prog.
87.13	5K	DAENTRY.BAS	Date Entry Program
87.14	8K	EFENTRY.BAS	Federal Tax Table Entry Program
87.15	5K	EPCUTS.BAS	Payroll Cutoff and % Entry Prog.
87.16	3K	EPLABELS.BAS	Empl. Mailing Label Printing Prog.
87.17	7K	EPTABS.BAS	Payroll Check Tab Entry Program
87.18	11K	ESENTRY.BAS	Cal. State Tax Table Entry Prog.
87.19	4K	FGALERT.BAS	Finished Goods Inventory Alert Printer
87.20	11K	FGENTRY.BAS	Finished Goods Inventory Entry Program
87.21	4K	FGRPT.BAS	Finished Goods Inventory Report Printer
87.22	3K	FGSORT.BAS	Finished Goods Entry Sort Prog.
87.23	5K	FMTENTRY.BAS	$ and % Format Entry Program
87.24	11K	GLHENTRY.BAS	G/L Heading Entry Prog.
87.25	3K	GLHSORT.BAS	G/L Heading Entry Sort Program
87.26	11K	GLSENTRY.BAS	G/L Subheading Entry Program
87.27	3K	GLSSORT.BAS	G/L Subheading Entry Sort Prog.
87.28	9K	MAENTRY.BAS	Mailing List Name Entry Program
87.29	3K	MALABELS.BAS	Mailing Label Printing Program
87.30	3K	MASORT.BAS	Mailing List Entry Sort Program
87.31	3K	MASTER4.BAS	Inventory Menu
87.32	2K	MASTER6.BAS	Mailing List Menu
87.33	3K	MASTER7.BAS	Initialization Routines Menu
87.34	1K	MASTER8.BAS	Periodic Maintenance Menu
87.35	1K	MASTER9.BAS	General Ledger Heading Menu
87.36	1K	MASTER15.BAS	Check Tabs and Cutoffs Menu
87.37	2K	MASTER16.BAS	Fixed Asset Accounting Menu
87.38	5K	MMAINT.BAS	Monthly File Maintenance Program
87.39	9K	NAMENTRY.BAS	Company Name Entry Program
87.40	3K	NMSORT.BAS	Company Name Entry Sort Prog.
87.41	6K	QMAINT.BAS	Quarterly File Maintenance Prog.
87.42	4K	RGALERT.BAS	Raw Goods Inventory Alert Printer
87.43	11K	RGENTRY.BAS	Raw Goods Inventory Entry Prog.
87.44	4K	RGRPT.BAS	Raw Goods Inventory Report Printer
87.45	3K	RGSORT.BAS	Raw Goods Entry Sort Program
87.46	7K	TAENTRY.BAS	General Ledger Tab Entry Program
87.47	3K	VPLABELS.BAS	Mailing Label Printing Program.
87.48	6K	YMAINT.BAS	Yearly File Maintenance Program

Volume 88

DESCRIPTION: BUSINESSMASTER II, Volume 3 of 5:
sample data files; payroll;

SUBMITTED BY: See above

NO.	SIZE	NAME	COMMENTS
	4K	-CATALOG.088	CONTENTS OF VOL. 088
	5K	U-G-FORM.LIB	Users Group Submission Form.
	2K	CRCKLIST.088	File of all file CRC's.
	2K	CRCK.COM	To check files on this disk.

Sample Data Files

88.1	6K	CG.	Cost of goods sold file.
88.2	1K	CGSIZE.	# in use + max size of above
88.3	1K	CHK.	?? next check # to write??
88.4	9K	CR.	Customer receivables file
88.5	1K	CRSIZE.	# in use + max size of CR file
88.6	1K	DATE.	Date information
88.7	4K	EDEP.	Federal deposit record
88.8	1K	EDEPSIZE.	# in use + max size of EDEP file
88.9	1K	EF.	Federal Withholding Tables
88.10	11K	EP.	Consolidated employee payroll
88.12	1K	EPSIZE.	# in use + max size of EP file
88.11	1K	EPC.	Payroll cutoff amounts
88.13	1K	EPT.	Payroll Check Printing Tabs
88.14	1K	ES.	California Withholding Tables
88.15	6K	FG.	Finished goods inventory
88.16	1K	FGSIZE.	# in use + max size of FG file
88.17	14K	GL.	General ledger file
88.29	1K	GLSIZE.	# in use + max size of GL file
88.18	6K	GLCD.	Check disbursements
88.19	1K	GLCDSIZE.	# in use + max size of GLCD file
88.20	2K	GLCK.	Check receipts
88.21	1K	GLCKSIZE.	# in use + max size of GLCK file
88.22	1K	GLF.	General ledger formats
88.23	2K	GLH.	General ledger heading file
88.24	1K	GLHSIZE.	# in use + max size of GLH file
88.25	13K	GLJO.	General journal gljo
88.26	1K	GLJOSIZE.	# in use + max size of GLJO file
88.27	1K	GLREF.	General Ledger Account # Reference File
88.28	4K	GLS.	general ledger subheading file
88.30	1K	GLSSIZE.	# in use + max size of GLS file
88.31	1K	GLT.	General ledger report tabs
88.32	1K	INV.	Invoice # File
88.33	9K	IR.	Invoice register
88.34	1K	IRSIZE.	# in use + max size of IR file
88.35	8K	MA0.	Mailing list file
88.36	1K	MA0SIZE.	# in use + max size of MA0 file
88.37	1K	NM.	Company Name File
88.38	1K	NMSIZE.	# in use + max size of NM file
88.39	3K	PO.	Purchase order register
88.41	1K	POSIZE.	# in use + max size of PO file
88.40	1K	POE.	Purchase order numbers
88.42	3K	TM.	Time card record
88.43	5K	VP.	Vendor payables file
88.44	1K	VPSIZE.	# in use + max size of VP file

Common Programs

These would not fit on this volume because of 64 directory entry limit. Get them from another volume (87, 89, 90).

1K	ALL.BAS	Include FORMAT and CONTROL
4K	BIZMII.BAS	Master Menu BUSINESSMASTER II. (bizmii)
2K	CHECK.BAS	Check for needed files
1K	CONTROL.BAS	Set up control characters for term.
1K	FORMAT.BAS	Set common and formats.
1K	READFILE.BAS	Read name, date, formats and tabs.

Payroll Programs

	1K	COMPILE.SUB	To compile all following programs
88.45	4K	EPCHECKS.BAS	Payroll Check Printer
88.46	1K	EPCLEAR.BAS	Employee payroll clearing program
88.47	8K	EPDPOSIT.BAS	Federal Tax Deposit Entry Program
88.48	14K	EPENTRY.BAS	Employee Payroll Rec. Entry Prog.
88.49	6K	EPJOPOST.BAS	Payroll Journal Posting Program
88.50	9K	EPJOTRAN.BAS	Payroll Calculation Program
88.51	3K	EPLIST.BAS	Payroll File Listing Program
88.52	5K	EPSORT.BAS	Employee Info Entry Sort Program
88.53	5K	EPSUMARY.BAS	Payroll Register Printer
88.54	5K	FED941PR.BAS	Federal Form 941 Quarterly Tax Return Printer

Volume 89

DESCRIPTION: BUSINESSMASTER II, Volume 4 of 5: purchase order/payables; order entry/receivables;

SUBMITTED BY: See above

NO.	SIZE	NAME	COMMENTS
	2K	-CATALOG.089	CONTENTS OF VOL. 089
	5K	U-G-FORM.LIB	Users Group Submission Form.
	2K	CRCKLIST.089	File of all file CRC's.
	2K	CRCK.COM	to check files on this disk.
	1K	COMPILE.SUB	To compile all following programs
89.1	1K	ALL.BAS	Include FORMAT and CONTROL
89.2	4K	BIZMII.BAS	Master Menu BUSINESSMASTER II. (bizmii)
89.3	2K	CHECK.BAS	Check for needed files
89.4	1K	CONTROL.BAS	Set up control characters for term.
89.5	1K	FORMAT.BAS	Set common and formats.
89.6	1K	READFILE.BAS	Read name, date, formats and tabs.
89.7	12K	CRENTRY.BAS	Customer Information Entry Prog.
89.8	7K	CRFMINV.BAS	Formatted Invoice Printing Prog.
89.9	8K	CRJOLIST.BAS	Invoice Register Printing Program
89.10	5K	CRJOPOST.BAS	Accounts Receivable Posting Prog.
89.11	13K	CRJOTRAN.BAS	Order/Invoice Entry Program
89.12	3K	CRLABELS.BAS	Customer Mlg Label Printing Prog.
89.13	7K	CRPPINV.BAS	Preprinted Invoice Printing Prog.
89.14	4K	CRRPT.BAS	Aged Accounts Receivable Statement Printer
89.15	3K	CRSORT.BAS	Customer Information Entry Sort Program
89.16	1K	INVENT.BAS	Invoice Number Sequencing Entry Program
89.17	3K	MASTER2.BAS	Accts Rcvable/Customer Menu
89.18	3K	MASTER3.BAS	Accounts Payable/Vendor Menu
89.19	1K	POENT.BAS	P.O. # Sequencing Entry Prog.
89.20	12K	VPENTRY.BAS	Vendor Information Entry Program
89.21	7K	VPFMPO.BAS	Formatted P.O. Printing Prog.
89.22	8K	VPJOLIST.BAS	P.O. Register Printing Prog.
89.23	5K	VPJOPOST.BAS	Accounts Payable Posting Program
89.24	13K	VPJOTRAN.BAS	Purchase Order Entry Program
89.25	3K	VPLABELS.BAS	Mailing Label Printing Program
89.26	7K	VPPPPO.BAS	Preprinted P.O. Printing Prog.
89.27	4K	VPRPT.BAS	Aged Accounts Payable Statement Printer
89.28	3K	VPSORT.BAS	Vendor Info. Entry Sort Prog.

Note: ALL, FORMAT, and CONTROL are %include files.

Volume 90

DESCRIPTION: BUSINESSMASTER II, Volume 5 of 5: general ledger;

SUBMITTED BY: See above

NO.	SIZE	NAME	COMMENTS
	3K	-CATALOG.090	CONTENTS OF VOL. 090
	5K	U-G-FORM.LIB	Users Group Submission Form.
	1K	CRCKLIST.090	File of all file CRC's.
	2K	CRCK.COM	to check files on this disk.
	1K	COMPILE.SUB	To compile all following programs
90.1	1K	ALL.BAS	Include FORMAT and CONTROL
90.2	4K	BIZMII.BAS	Master Menu BUSINESSMASTER II. (bizmii)
90.3	2K	CHECK.BAS	Check for needed files
90.4	1K	CONTROL.BAS	Set up control characters for term.

(continued next page)

90.5	1K	FORMAT.BAS	Set common and formats.
90.6	1K	READFILE.BAS	Read name, date, formats and tabs.
90.7	11K	GCINCOME.BAS	Comparative Income Statement Printer
90.8	4K	GDCHECKS.BAS	General Check Printer
90.9	5K	GLBALSHT.BAS	General Ledger Balance Sheet Printer
90.10	3K	GLBUDGET.BAS	General Ledger Budget Analysis Printer
90.11	9K	GLCDPOST.BAS	Check Disbrsmt Posting Prog.
90.12	4K	GLCDSORT.BAS	Check Disbrsmts Entry Sort Prog.
90.13	15K	GLCDTRAN.BAS	Check Disbrsmts Jrnl Entry Prog.
90.14	4K	GLCHART.BAS	G/L Chart of Accounts Printer
90.15	9K	GLCKPOST.BAS	Check Receipt Posting Program
90.16	4K	GLCKSORT.BAS	Check Disbrsmts Entry Sort Prog.
90.17	14K	GLCKTRAN.BAS	Check Rcpts Journal Trans. Entry Program
90.18	1K	GLCLEAR.BAS	Clear all $ values in the G/L file
90.19	11K	GLENTRY.BAS	G/L Entry Program
90.20	10K	GLINCOME.BAS	Income Statement Printer
90.21	9K	GLJOPOST.BAS	General Journal Posting Program
90.22	4K	GLJOSORT.BAS	General Journal Sorting Program
90.23	13K	GLJOTRAN.BAS	General Journal Trans. Entry Program
90.24	3K	GLSORT.BAS	G/L Entry Sort Program
90.25	6K	GLTBAL.BAS	G/L Trial Balance Printer
90.26	3K	MASTER1.BAS	General Ledger/General Journal Menu
90.27	2K	MASTER11.BAS	General Journal Menu
90.28	1K	MASTER12.BAS	Check-Cash Rcpts-Disbrsmts Menu

Note: ALL, CONTROL, and FORMAT are %include files.

Abstracts

BUSINESSMASTER II is a general business software package for CBASIC2, occupying CPMUG volumes 86-90, encompassing, by volume:

86 documentation;
87 initialization, startup, modification and maintenance; inventory/fixed asset accounts; mailing list;
88 sample data files; payroll;
89 order entry/receivables; purchase order/payables;
90 general ledger;

There were seven single density disks originally contributed: five program disks one documentation disk, and one sample file disk. I combined these seven disks to produce five CPMUG volumes. Thus not all references in the various documentation will directly apply disk by disk, but all the files submitted are included.

Only the files "CRJOSORT.BAS", "VPJOSORT.BAS", and the submit files for copying, XREFing, PIPping, and compiling with printing are in the documentation but not included in this collection. The contributor felt the two missing ".BAS" programs to be unnecessary. Also a minor bit of REM documentation seems to be missing off of the front of GLJOSORT.BAS.

I generated a compile submit file for each disk. The submit files have the form:

```
$1 $2filename.typ $3 $4
```

Thus to compile them from the A: disk to the B: disk, type:

```
submit compile CBASIC2 A: B: $B
```

There are six .BAS files which are on every disk: ALL, FORMAT, CONTROL, CHECK, READFILE, AND BIZMII.

The remainder of this abstract is the press release officially releasing BUSINESSMASTER II into the public domain.

Ward Christensen

BUSINESSMASTER II Released into Public Domain

For the protection of those who have purchased a package of General Business Software known as BUSINESSMASTER II or Visaccount (which was, as of March 1982, at least 95% derivative of BUSINESSMASTER II) from either OEM Software (a Star Computer Systems company), Computer Services Corporation of America, 800-Software or other dealers OEM Software has established, we are hereby declaring this old version of BUSINESSMASTER II to be in the Public Domain. The Copyright notices were removed by OEM Software and the package was then sold to over 400 dealers for $375.

One of many dealers purchasing the package was Computer Services Corporation of America. Buck Lindsey CEO of Computer Services Corporation of America advised me last year that he felt the package was in the public domain and whether it was then or not, it is now.

Computer Services Corporation of America made some minor changes to the package and renamed it Visaccount. Additionally they made 800-Software a discount dealer.

We are placing this package in the Public Domain to clear up a problem that has been generated for literally thousands of people who purchased this package from one source or another and currently do not know where they stand.

For those End-Users who are not satisfied with their Version of BUSINESSMASTER II or Visaccount we will exchange your package for the new and much improved BUSINESSMASTER II+ for $100. Simply mail your original diskettes (OEM or Visaccount) and license to us. We will put the new package on your diskettes with new labels, a new licensing agreement and a new manual and return it to you.

For dealers who purchased BUSINESSMASTER II from OEM software, you may continue to do as you wish with the package but we cannot upgrade your package to BUSINESSMASTER II+. However we do offer dealer discounts on our newer packages.

We have offered the package to The CP/M Users Group for inclusion in their library on the understanding that it may be used in any way, either commercial or non-commercial by anyone who chooses to use it provided credit is given for such use.

We will continue to supply BUSINESSMASTER II+ to end-users for $159 and BUSINESSMASTER Plus with fully-formatted fill in the blanks screen and B Tree Indexing for $289. These packages are compilable with minor changes under CB-80.

(signed)
Bud Aaron
BUSINESSMASTER

Feature

A Review Of Dataflex

Steve Patchen

Data Access Corporation
4221 Ponce De Leon Blvd.
Coral Gables, Florida 33146
List Price: $750
Demo Package: $100

Dataflex is a database system including entry, reporting and other utilities for application development. The data management routines are also available optionally as MTPascal libraries. The source code for the Entry and the Report routines are included with the libraries. The system is therefore available as a fairly complete application development environment. The availability of the libraries and source code permits custom programming, useful in overcoming unusual problems encountered with some applications. Without programming, implementation using the existing entry and report facilities is still too complicated for beginners and non-programmers. Even experienced programmers will find their initial learning experiences difficult. However, when one becomes familiar with the system, entry screens and reports can be implemented fairly quickly.

The data structures utilized by the system are Codasyl network type parent-to-child relationships, one way associations - not two-way mappings. Thus, a one-to-many relationship in one direction does not imply a many-to-one relationship in the other direction. Many-to-many relationships require an additional root file, providing the multiple links to the two files. It might be possible to provide fully relational mappings by writing additional Pascal programs which bypass some of the library functions, but this restriction does not cause trouble in most cases.

The second report of the example application attempts to implement a cyclic relationship (see Hubbard page 36). On the recommendation of Data Access, I created a second file definition for the first file (30 and 32) to force the system to use two buffers for the file. The way the system works you cannot force the first file definition buffer to abandon the original record and find records by references from the second file (31), field (4). Relationships are explicitly defined in the file definitions, as are the ISAM indices.

The example application which I developed to exercise the Dataflex system is a simple reference card entry and reporting system. I utilized the cards to describe the Dataflex components, so you should read the example report as required. Each card consists of a title, a category, a date, twelve lines of text and a list of references. The parts are two files, an entry routine and two report routines. The first report dumps all the cards in the files and prints each on an 8 1/2 by 11 inch page. The second report attempts to print all the cards referenced by a single selected card. Using the first report, I dumped the cards into the article, then removed the blank lines and did a little more editing, so the cards do not look exactly as they would if printed by Dataflex. The second report was not tested because I ran out of memory. (Dataflex requires a TPA of 51k. My system is a 64k Z80 with Morrow

```
                    CARD FILE REPORT

PAGE____        CONCEPT CARDS           DATE: ___/___/___
_____________________________________   UPDATED: ___/___/___
         CATEGORY: ________________
______@________________________________________________
_______@_______________________________________________
________@______________________________________________
_________@_____________________________________________
__________@____________________________________________
___________@___________________________________________
____________@__________________________________________
_____________@_________________________________________
______________@________________________________________
_______________@_______________________________________
________________@______________________________________
_________________@_____________________________________
REFERENCES:
 NO.   CATEGORY           CARD TITLE
 ____  ___________  ________@_______________________

_______________________________________________________
/*
1 1 30 1 31 1
55
1
0 0 0 0
1 17 1 19
20 22 20 20
30 30 21 22
0 0 0 0

1 2 0 0 1
2 0 1 2 2
3 0 30 1 3
4 0 30 2 4
5 0 30 3 5
6 0 30 4 6
7 0 30 5 7
8 0 30 6 8
9 0 30 7 9
10 0 30 8 10
11 0 30 9 11
12 0 30 10 12
13 0 30 11 13
14 0 30 12 14
15 0 30 13 15
16 0 30 14 16
17 0 30 15 17

20 0 31 2 18
21 0 31 3 19
22 0 31 4 20

30 12 0 0 57

999
/*
```

(continued next page)

Designs drives and a Winchester. The Winchester requires about 4.5k worth of drivers. This puts my BDOS at D600H which is only a 53.5k TPA. Having only 2.5k above the minimum does not allow me to open the third file I needed.)

The reports and entry screen forms have similar formats. A mask for the report or screen is followed by a specification program, consisting of some lines of declarations followed by numbered lines of code. The programs are composed almost entirely of lines of numbers, except where one is differentiating between screen window references and file field references. (W5 references the 5th screen window and F1,5 references the 5th field of file 1.) This primitive method of programming makes it very difficult to read the programs and even harder to create them. It takes a considerable amount of time to learn the numeric codes before a programmer can get started with even the simplest application. The programs are usually short, so it is nevertheless possible to become proficient at creating them.

The body of the program basically specifies what action should be taken at each window. In addition, conditional branching is possible to anywhere within the window processing sequence. Windows can be modified by the operator or by expressions using windows and file fields. The screen form also allows for up to six dedicated subroutines. If present, one is activated upon entry to the routine, one by the delete function key, one by the save function key or the end of screen save, one prior to completing a requested screen clear, one before exiting the program and one is reserved for the user function key.

The screen mask consists of literal strings and fixed length field definitions, with three picture formats to match the three data types. A string consisting of underline characters only defines an ASCII string field. If a period appears anywhere in the field, the field is defined as a numeric type. The picture '____/____/____' or '____/____/________' defines a date format. The same entry format is used for locating, editing, creating and deleting records. A find function key must be depressed in a key field containing the key for the record required, in order to locate and display a record for editing or deletion. Creating a record is performed by clearing the screen fields and entering new information. The edited or new record is saved by pressing the save function key or by responding to the prompt issued when passing the last screen field.

The behavior of the screen is at first confusing. Operations of most of the special function keys are straightforward. The space forward and back, the delete and insert character and the key to clear all windows act as expected (almost).

The keys I found most difficult were those which position the file records. If you are in a window with a field which is a key to a record and request a *find*, the system locates the record with that key (if it is not already current). Likewise, if you are on a key to the primary file for the screen files and request a *superfind* all the records referenced are located. If you are in a window which does not contain an index field you get an error message; if you are not in a primary index window results are unpredictable.

The previous and next record functions act differently, depending upon where the cursor is and whether a *find* or *superfind* function has been executed. The previous and next record functions move you from one card to the next if you are in the title window. If you are in the reference number

EXAMPLE REPORT OF CONCEPT CARDS

```
PAGE 1          CONCEPT CARDS                    DATE: 9/ 2/82

DATAFLEX_DESCRIPTION                          UPDATED: 9/ 2/82
                     CATEGORY: ARTICLE
The DATAFLEX system is composed of File definition routines, an
Entry procedure, a Report procedure, a Query procedure, a menu
utility and some utility programs. In addition there is a terminal con-
figuration routine.
   The system is written in MTPascal. The source listings are avail-
able separately. A section of the manual is devoted to the description
of these Pascal modules and programs.
   The manual and diskettes contain sample data files and Entry and
Report forms to illustrate the creation of applications. The manual in-
troduction has a discussion of system concepts and a user level in-
troduction. The rest of the manual contains information required to
implement application systems. There are appendices covering er-
ror messages, file requirements and expressions.
REFERENCES:
 NO.  CATEGORY          CARD TITLE
  1     ARTICLE   DATAFLEX_ENTRY
  2     ARTICLE   DATAFLEX_REPORTS
  3     ARTICLE   DATAFLEX_QUERY
  4     ARTICLE   DATAFLEX_FILES
  5     ARTICLE   DATAFLEX_MANUAL
  6     ARTICLE   DATAFLEX_UTILITIES
  7     ARTICLE   DATAFLEX_MENUS
  8     ARTICLE   DATAFLEX_INSTALLATION

PAGE 2          CONCEPT CARDS                    DATE: 9/ 2/82

DATAFLEX_ENTRY                                  UPDATED: 9/2/82
                     CATEGORY: ARTICLE
The entry procedure consists of a screen format and the necessary
data files. The screen format is created with a word processor. It con-
sists of a mask for the screen display exactly as it will appear on the
screen with no data entered. This mask is followed by a specification
program written in lines of numbers. The language syntax is not dif-
ficult, but having to write programs by numbers is a regression to the
cave man days of programming.
   Errors in the program are reported on the bottom line. Frequently
they only flash on the screen for an instant. I found myself having to
develop ways of getting the error to repeat enough to read the
message. Although the syntax is not difficult, the structure of the
system is very hard to grasp. I do not think anyone without a great
deal of experience will be able to deal with the requirements of
developing applications.
REFERENCES:
 NO.  CATEGORY          CARD TITLE
  1     ARTICLE   SAMPLE_APPLICATION_DESCRIPTION
  2     ARTICLE   SAMPLE_ENTRY_PROGRAM

PAGE 3          CONCEPT CARDS                    DATE: 9/ 2/82

DATAFLEX_FILES                                UPDATED: 9/ 2/82
                     CATEGORY: ARTICLE
File creation and maintenance are handled by one routine. It has en-
tries to create files, define or edit fields, create or change indices, list
or print the definitions and to erase the data in a file. There is also a
facility to create single file images from a screen definition. When you
add or delete fields to a file with data in it the system automatically
repositions the fields in each record. Once when I made some exten-
sive changes I lost my data. Therefore, you should plan ahead when
changing the file structures.

REFERENCES:
 NO.  CATEGORY          CARD TITLE
  1     ARTICLE   FILE_DEFINITION
  2     ARTICLE   FILE_INDICES
```

window you move back and forth between the first and last references for the card you are in. If you attempt to exceed the existing references, an error message is displayed and scanning is inhibited until you execute a *find* or a *superfind*. A *superfind* allows the card boundaries to be exceeded and the card information changes as you pass from one card to the next. If you bump into one end or the other, you again get stuck until you issue a *find* or *superfind*.

There is a utility to create a file definition from a screen form, but it only works for single file entry screens. File definition is at the physical level. Conceptual and logical models must be constructed outside the system before implementation is attempted. Nevertheless, Codasyl network models could be mapped into Dataflex structures.

The menu system seems to run completely independently of the other modules. It is loaded with CP/M-80 command lines or 'MENU menuname' commands. I found the menus easy to understand and use, yet powerful. A typical command to run the screen entry would be 'ENTER CARDS'. This could be loaded into a menu selection or executed directly from CP/M. The menu system is an excellent integration tool. This and the Codasyl-like structures provide assurance that many application structures can be implemented.

There is no integral backup system, so you must use PIP or provide your own. A menu entry prompts for a source and destination file and calls PIP with the parameters entered. Lists of files and file type uses are provided in the appendices to help you decide which files have to be copied.

The most bothersome aspects of using Dataflex were writing programs as strings of numbers, and running out of memory for a simple task like the second report. The memory problem could be overcome by running Dataflex on one of the 16-bit processor machines it is available for; these have a larger memory address space. Possibly a feature could be provided to allow bank switching of extra memory for additional buffer space in 8-bit machines. The only solution to the unreadable program problem I can think of is a preprocessor which accepts symbolic labels and operator mnemonics. Otherwise the system is useful for implementing a wide range of applications.

References:

Hubbard, G.U. 'Computer Assisted Data Base Design', Van Nostrand Reinhold, NY, 1981.

Date, C.J. 'An Introduction to Database Systems', 2nd. Ed., Addison Wesley, Reading, Mass. 1977

Lifelines, The Software Magazine:

"The Software Evaluation Group", Vol I No.4

"How to Use a Data Management System", Vol I No.5

"The Software Evaluation Group Review Format", co-authored with E. Paulette, Vol. I No. 5

"Introduction to Data Management Systems" by J. Lehman & T. Berla, Vol. I. No. 6

"The Software Evaluation Group: Business Application Problem Definitions", co-authored by E. Paulette, Vol. I No. 7

"A Review of the Condor Database System", co-authored by E. Paulette, Vol. I No. 11

"The Software Evaluation Group: SELECTOR IV", co-authored by T. Berla, Vol. II No. 1

"Criteria For Evaluating Application Development Software", Vol. III No.1

```
PAGE  4          CONCEPT CARDS                    DATE: 9/  2/82

DATAFLEX__MANUAL                               UPDATED: 9/  2/82
                        CATEGORY: ARTICLE

The manual does not contain an index. The table of contents covers
each sub-section but I spent a long time trying to relocate particular
terms or explanations. There are summary tables but many are in in-
dividual sections rather than in an appendix. Except for general
operator instructions, understanding and using the system requires
the experience of a programmer. The manual is over 200 pages and
contains a lot of detail. The explanations are clear, but often impor-
tant implications are not discussed. I found that I had to spend a lot of
time learning DATAFLEX before I could do even the simple applica-
tion attempted for this article.

REFERENCES:
 NO.  CATEGORY          CARD TITLE
  1     ARTICLE    DATAFLEX__MANUAL

PAGE  5          CONCEPT CARDS                    DATE: 9/  2/82

DATAFLEX__MENUS                                UPDATED: 9/  2/82
                        CATEGORY: ARTICLE
The menu system can route to other menus, can perform flex opera-
tions or CP/M commands by selection. The menu editor allows you
to create, display and delete menus. The direct commands include
substitutes for the CP/M DIR, ERA, TYPE, USER and SUBMIT. In ad-
dition NAME does the REN function and RESET does a warm boot.
There is also a PAUSE and a CLS to clear the screen. You can re-
quire a password to gain access to individual menu selections.
Parameters can be introduced to menu selections by providing
operator prompts. The menu system can be configured to autoload.

REFERENCES:
 NO.  CATEGORY          CARD TITLE
  1     ARTICLE    none

PAGE  6          CONCEPT CARDS                    DATE: 9/  2/82

DATAFLEX__QUERY                                UPDATED: 9/  2/82
                        CATEGORY: ARTICLE

The Query routine allows ad hoc and selective listing of records and
fields, one file at a time. The operator is led through a series of ques-
tions to fill out a query specification. The query can be sent to the
printer, terminal or a file. This routine allows versatile interaction with
individual files but not queries or reports on related files. The report
routine should be used for reports from related files.

REFERENCES:
 NO.  CATEGORY          CARD TITLE
  1     ARTICLE    DATAFLEX__QUERY

PAGE  7          CONCEPT CARDS                    DATE: 9/  2/82

DATAFLEX__REPORTS                              UPDATED: 9/  2/82
                        CATEGORY: ARTICLE

The report procedure is composed of parts similar to the entry. The
report program is more complicated, consisting of Header,
Subheader, Body, Subtotal and Total sections. It did not take me as
long to implement the report as it did the entry screen, due in part to
my having gained experience with the system and in part to my call-
ing Data Access for help when I didn't understand why it wouldn't
work. The system puts everything into memory, minimizing disk ac-
cesses. However, this limits the complexity of the report and the
number of files you can have active in any report. Updating of files is
conducted while executing a report routine. As many files as
memory allows can be used in the update.

REFERENCES:
 NO.  CATEGORY          CARD TITLE
  1     ARTICLE    SAMPLE__APPLICATION REPORT
```

→

```
PAGE  8              CONCEPT CARDS              DATE: 9/  2/82

DATAFLEX_UTILITIES                           UPDATED: 9/  2/82
                         CATEGORY: ARTICLE

There are three utilities provided in addition to the other routines. The
Re-Index routine rebuilds damaged or lost indices. The Free List
routine rebuilds the free list of deleted records in a file. It is able to
recover files with garbaged records by deleting them. The Read
routine is able to load an ASCII file with one field per line to a
database file.

REFERENCES:
 NO.  CATEGORY              CARD TITLE
  1      ARTICLE     none
```

```
PAGE  9              CONCEPT CARDS              DATE: 9/  2/82

DATAFLEX_INSTALLATION                        UPDATED: 9/  2/82
                         CATEGORY: ARTICLE

Installation is implemented in four parts. The first asks the operator
some questions which can be answered with a yes or no. They
establish whether the system is for multi-users and if the return key is
required to leave a field. Part two asks the operator to respond with
keys to be used for several special functions. The third and fourth
parts require the operator to enter the decimal equivalent of number
sequences sent to the terminal to control it and the cursor. The menu
for the installation program has options to view the configuration on
the screen or to print it.

REFERENCES:
 NO.  CATEGORY              CARD TITLE
  1                  none
```

ENTRY SCREEN SPECIFICATION

NAME:____________________ DATE:____/____/____

TYPE:____________________

____

____

____

____

REFERENCES:

____

____

```
/*
*1  30  31  32
0 0 0 0 0
1  34  32  2  2  :F1,2          ; DATE THE FILE
2  40  30  1  1                 ; NAME
3  145  30  0  11               ; SKIP DISPLAY IF SAME RECORD
#  33  30  3  3
#  33  30  4  4
#  33  30  5  5
#  33  30  6  6
#  33  30  7  7
#  33  30  8  8
#  33  30  9  9
11  8  31  2  10
12  145  31  0  20              ; SKIP IF SAME RECORD
13  1  31  3  11
#  1  31  4  12
#  1  31  5  13
#  1  31  6  14
#  1  31  7  15
#  1  31  8  16
#  1  31  9  17
20  8  32  3  18
21  1  32  4  19
#  255
```

CARD REFERENCE REPORT

PAGE____ CONCEPT CARDS DATE:____/____/____

TITLE CARD__________@______________

REFERENCE CARD______________ NO.________

UPDATED:____/____/____ CATEGORY:______________

______@______________________________

________@____________________________

__________@__________________________

____________@________________________

______________@______________________

________________@____________________

__________________@__________________

____________________@________________

______________________@______________

________________________@____________

__________________________@__________

____________________________@________

```
/*
1  1  30  1  32  1  31  1
55
1
0  0  0  0
1  3  1  5
4  19  5  20
30  30  21  22
0  0  0  0

1  2  0  0  1                         ; TODAY'S DATE
2  0  1  2  2
3  0  30  1  3

4  0  31  4  4
5  0  31  2  5
6  0  32  2  6
7  0  32  3  7
8  0  32  4  8
9  0  32  5  9
10  0  32  6  10
11  0  32  7  11
12  0  32  8  12
13  0  32  9  13
14  0  32  10  14
15  0  32  11  15
16  0  32  12  16
17  0  32  13  17
18  0  32  14  18
19  0  32  15  19

30  12  0  0  57

999
& ENTER TITLE FOR CARD REFERENCES YOU WANT:
?  12  31  1  $&1
/*
```

TABLE R5
APPLICATION GENERATION SYSTEMS

I. APPLICATION SUITABILITY

1. Does the method required for specifying applications reflect an understandable and logical model of the domain of applications for which it is intended?

   Although problem specification must be done before using Dataflex the data model used is close to the Codasyl network model.

TABLE R5 (cont)

2. Can the application be completely specified?

   A wide range of applications could be specified in a manner that could be implemented by this system.

3. Is the implemented system testable against the specification for the system or vice versa?

   Implementation is not always straightforward and might not reflect the application directly.

4. Does the development system make it easy to extend and rework the specification and implementation?

   Reworking implementation is not too difficult.

II. IMPLEMENTATION SUITABILITY

1. Are the restraints and limitations of the implementation environment made clear to the designer?

   Logical limitations are only partially explained in the manual.

2. Are tools for system implementation complete or are other independent tools required?

   I had to provide my own editor, but they have an editor available.

3. Is the implementation environment extensible to include new components or components from other systems?

   Yes, libraries are available for the data management routines and source code for the entry and report routines are available for use with MTPascal.

III. USER/DESIGNER SUITABILITY

1. Are the user interfaces developed by system and those used to develop application understandable from terms of tasks to be performed or do unrelated details obscure the operation?

   There are only a few confusing details that the operator has to face, but the designer has many confusing details to face.

2. Does user feel in complete control of system or do obtuse messages and unexplained operations leave him in confusion or frustration?

   There are a few frustrations.

3. Does the system seem to have been designed with psychological criteria for short term memory, closure of tasks, response time and user control in mind?

   Maybe, but not consistently.

IV. MACHINE SUITABILITY

1. Are limitations imposed by the machine environment understandable in terms of application limitations and are application requirements translatable to machine requirements?

   INDIRECTLY understandable. Designer is notified of available memory at key places.

2. Are any provisions made in the development system to allow optimization in different machine environments?

   NO

3. Is it possible to extend the machine environment without major changes to applications already implemented?

   YES

```
        FILE DEFINITION LISTING FOR FILE #30
****************************************************
        FILE ROOT NAME    = CONCEPTS
        USER DISPLAY NAME = CONCEPTS
        SHORT NAME        = CONCEPTS
****************************************************
        RECORD LENGTH = 1024  (USED = 986)
        MAX NUMBER OF RECORDS = 6000  (USED = 9)
        DELETED SPACE IS REUSED
        MULTI-USER RE-READ ACTIVE
****************************************************
FIELD   FIELD   FIELD   FIELD  DEC  MAIN  RELATES--TO
NMBR    OFFSET  LEN      TYPE  PTS  INDEX FILE  FIELD
-----   ------  -----   -----  ---  ----- ----  -----
   1         1     32  ASCII           1     0    0     TITLE
   2        33      3  DATE            0     0    0     DATE
   3        36     15  ASCII           0     0    0     TYPE
   4        51     78  ASCII           0     0    0     LINE1
   5       129     78  ASCII           0     0    0     LINE2
   6       207     78  ASCII           0     0    0     LINE3
   7       285     78  ASCII           0     0    0     LINE4
   8       363     78  ASCII           0     0    0     LINE5
   9       441     78  ASCII           0     0    0     LINE6
  10       519     78  ASCII           0     0    0     LINE7
  11       597     78  ASCII           0     0    0     LINE8
  12       675     78  ASCII           0     0    0     LINE9
  13       753     78  ASCII           0     0    0     LINE10
  14       831     78  ASCII           0     0    0     LINE11
  15       909     78  ASCII           0     0    0     LINE12

INDEX 1:  FIELD SEGMENTS:  <1>
```

```
        FILE DEFINITION LISTING FOR FILE #31
****************************************************
        FILE ROOT NAME    = CARDS
        USER DISPLAY NAME = CONCEPT_CARDS
        SHORT NAME        = CARDS
****************************************************
        RECORD LENGTH = 512  (USED = 81)
        MAX NUMBER OF RECORDS = 6000  (USED = 23)
        DELETED SPACE IS REUSED
        MULTI-USER RE-READ ACTIVE
****************************************************
FIELD   FIELD   FIELD   FIELD  DEC  MAIN  RELATES--TO
NMBR    OFFSET  LEN      TYPE  PTS  INDEX FILE  FIELD
-----   ------  -----   -----  ---  ----- ----  -----
   1         1     32  ASCII           1    30    1     TITLE
   2        33      2  NUMERIC   0     1     0    0     REFERENCE_NUM
   3        35     15  ASCII           0     0    0     TYPE
   4        50     32  ASCII           2    32    1     CARD_REFERENCE

INDEX 1:  FIELD SEGMENTS:  <1>    <2>
INDEX 2:  FIELD SEGMENTS:  <4>    <0>
```

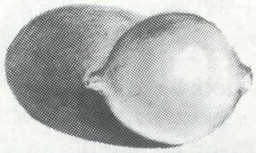


TABLE R2
Qualitative Factors

	Rating*
Documentation	
organization for learning	4
organization for reference	4
readability	6
includes all needed information	5
Ease of use	
initial start up	6
conversion of external data	4
application implementation	5
operator use	5
Error Recovery	
from input error	6
restart from interruption	6
from data media damage	6
Support	
for initial start up	6
for system improvement	†

† Additional library and source code availability make system improvement possible.

* Ratings in this table will be in a 1-7 scale where:
1 = clearly unacceptable for normal use
4 = good enough to serve for most situations
7 = excellent, powerful, or very easy depending on the category

(continued next page)

TABLE R1
Facts & Figures

Package or Version name:
DATAFLEX VERSION 1.6

Price:

List price	$750
Demo Package	$100

Systems available for:
CP/M-80, Z80, 8085, 8088, 8086 CP/M-86, MSDOS, TURBODOS, IBM, XEROX, HP, IMS, TELEVIDEO, ALTOS, TRS-80 II, S-100
requires cursor addressable terminal

Required supporting software:
An editor; one is discussed in the manual but was not provided with the demo package
MTPascal is required if the libraries are purchased

Memory requirements:
51k of transient program area
>54k TPA for example application

Diskette capacity required:
300k

Utility programs provided:
Routines to re-cover indices and files with damaged records are provided
As additional purchase option: MT Pascal routine libraries and source for the entry and report routine

Record size & type limits:

125 files max.	255 fields per file
5 indices/file	8M bytes per file
64k records/file	4k bytes per record

as memory size permits!!!!!!

Portability:
should be limited only by media compatibility

User skill level required:
professional skills required for implementation of applications, operators will require training

System upgrade policy:
none mentioned

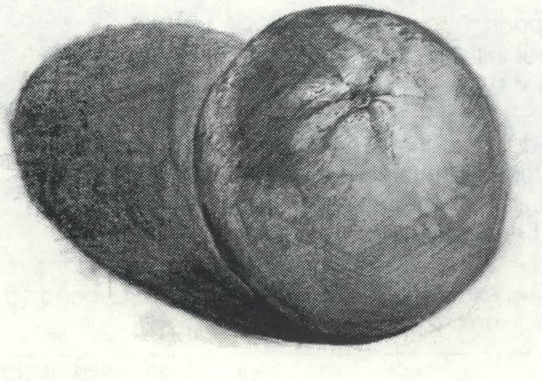


TABLE R3
Data Management Capabilities

A. Underlying Data Model

1. Data Types
   ASCII, NUMERIC(integer and fixed point) DATES
2. Relationships
   1:1,1:M,M:M all one direction associations

B. Functions Provided

1. a. Data dictionary maintenance
   A simple file dictionary is provided
   b. Data reorganization & conversion.
   Some field reorganization is provided data conversion is limited to one format.
2. a. Data entry and editing GOOD
   Multiple file entry is possible.
   b. Report generation GOOD
   Single file queries and multifile reports are provided
3. a. Data selection by predicate
   Selection of records by predicate is used in reports and queries.
   b. Data joining & relating multiple data sets
   Multiple file relations are provided joining is present only as file updating from multiple files.
   c. Calculations on data GOOD
   Including conditional calculation.
4. a. Data independent application interface
   Good independence is provided by the data management routine libraries

TABLE R6
APPLICATION DEVELOPMENT FACILITIES

FUNCTIONAL PARTS	Completeness and Complexity of Facilities			
	Little or None	Some	Complete & Complex	Easily Complex
Individual Program Development	▨	▨	▨	1
Input Transactions	▨	▨	YES	▨
Data Management	▨	▨	2	▨
Reports & Queries	▨	▨	YES	▨
Integrated Systems	▨	▨	▨	3

NOTES

1) The available routine libraries and MTPascal interface make full and complex programmming available easily.

2) The database management facilities are good, but the data dictionary management is limited.

3) A menu system is provided to integrate applications with. Organization of larger systems would suffer from lack of a more complete data dictionary system.

Macros of the Month

Edited by Michael Olfe

How about a macro which automatically stamps your files with time, date, version number, and author when they are created or edited? If you have a hardware clock in your system, this month's macro from Bill Norris of Bronx, New York, will do the job nicely. The header on "IOPATCH.ASM" listed below was created by this macro. Additionally, the macro will allow you to call machine-language subroutines from within PMATE just as you would make CP/M system calls - by loading a register with the function number, and calling a single entry point. Mr. Norris has used such calls to fix a deficiency in PMATE - the lack of access to different user areas. The "IOPATCH.ASM" below allows saving the current user number and setting or restoring it. For example,

```
6q0 xm ; saves current user # in user variable 1
7q0 xm ; restores current user # from user variable 1
5q1 7q0 xm ; sets current user # to 5
```

Several interesting features of PMATE are demonstrated by these macros, namely

Use of "XM" to call an assembly-language subroutine
Use of "@P" to pass parameters to a subroutine
Use of macros as subroutines

Implementing The Time-Date Macros

1. You must have a clock/calendar in the system.
2. Edit "IOPATCH.ASM", inserting a routine to read the clock/calendar and store a time/date string in memory. The address of the routine must replace the monitor address. The example which follows is for the OKI MSM5832 chip on the Compupro System Support I board.
3. Assemble the new "IOPATCH.ASM", overlay the hex file onto a copy of "PMATE.COM", and save the new "PMATE.COM". For example:

```
A>ddt pmate.com
-iiopatch.hex
-r
↑C
A>save 91 pmate.com
```

4. Load the new "PMATE.COM", load the permanent macros below, delete any leading comments, copy to the permanent macro area, and test the ".f" macro to be sure your copy was accurate. If all is well, duplicate PMATE with "XDPMDATE.COM".

```
A>pmate
xipermacs.pma$$
a6k
qmc$$
.f$$
Edit : q.c<cr>
<assuming .f worked>
xk$xdpmate.com$xh$$
A>
```

New Version Of PMATE

Version 3.21 of PMATE is now available for CP/M-80, CP/M-86, PCDOS, and MSDOS. Some of the new features in this version:

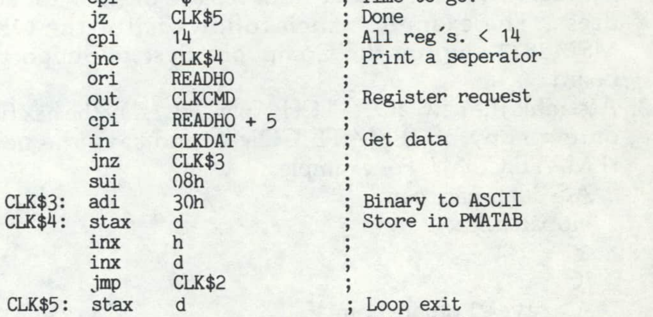
1. Control-S repeats the next keystroke four times or the number of times which follows.
2. New commands for auto-indent (a la UCSD Pascal editor):
   set auto-indent to cursor column
   increment auto-indent by 4 columns
   decrement auto-indent by 4 columns
   "nQ/" to set auto-indent to column n
3. Direct console I/O, with no instant command translation on input.
4. Improved configuration. It is no longer necessary to edit the CNF file if you have one of the terminals on the menu. There are also ten permanent macros defined as instant commands in the CNF file.

```
0=zero O=letter O
 ; File "IOPATCH.ASM"
 ;  Only relevant parts of file reproduced here
 ; This is iopatch for version 3.21, but
 ;  earlier versions should present no problems.
 ;**************************************************************
 ;*    Version: 006,   Time: 00:31:30,   Date: 09/10/82.
 ;**************************************************************
 ;*    Program: IOP.ASM              Author: Mike Aronson        *
 ;*    Modified by: Bill Norris                                  *
 ;*                                                             *
 ;*    1> Adds general purpose machine lang. interface.         *
 ;*    2> Uses Godbout System Support Board for time/date.      *
 ;**************************************************************

 ;THIS VERSION IS FOR MAC (Digital Research)
 ; and possibly ASM (also D.R.)
 ; ***** IMPOSSIBLE WITH M-80 (Microsoft) *****

MACSZ   equ     2048    ;SIZE OF PERMANENT MACRO AREA
 ; note: At least one of following two equates should be 0.
 ;       Allowable values are 0 or -1. (Both may be 0)
 ;       The value of setting both to 0 is that an IOPATCH.HEX
 ;       file may be ´passed around´ and used for other console
 ;       types. Just overlay the old PMATE (***  3.2x  ***).
MEMMAP  equ     0         ; -1 IF MEMORY MAPPED
CRT     equ     0         ; -1 for terminal
NODEV   equ     not (MEMMAP or CRT) ; to skip definition
UINIT   equ     5239H

        ; Equates used by Clock routine
cl$bas  equ     50h
clkcmd  equ     cl$bas+10
clkdat  equ     cl$bas+11
bdos    equ     5
read    equ     10h
hold    equ     40h
readho  equ     read+hold

        ORG 109H

                  ;JUMP VECTORS AND USER VARIABLES
UINITL: JMP UINIT         ;USER INITIIALIZATION
UEXIT:  RET               ;USER EXIT ROUTINE
        NOP
        NOP
CI:     JMP 0             ;CONSOLE INPUT VECTOR
CSTS:   JMP 0             ;CONSOLE STATUS VECTOR
COUT:   JMP 0             ;CONSOLE OUTPUT VECTOR
```

(continued next page)

```
LO:     JMP 0               ;LIST VECTOR
LSTS:   DB 0,0,0            ;LIST STATUS VECTOR
MONTR:  jmp     FERJOY  ;MONITOR VECTOR now via USER CALL #0.
MONITR  equ     0h      ; Or the entry point of your monitor.

 < part omitted >

        if NODEV
say hey;        Assembly for existing terminal definition
        ds      67
        endif

        DB 0

DELAY:  DB 100              ;DELAY TIME FOR QD COMMAND
     <part omitted>
                ; FOLLOWING VARIABLES CAN BE SET BY Q COMMANDS
                ; User variables
UVAR0:  DW 0    ; Mate stores USER CALL REQUEST # here

UVAR1:  DW 0
; Call #´s      ; Description of register contents
; 0             ; Jump to monitor
; 1, 2          ; Points to Time/Date string
; 3             ; Go to address stored here
; 4             ; Pointer to string to copy to UBUFF
;                 -  string ends w/^Z, max. length=80 chars
; 5             ; Pointer to location to overwrite from UBUFF
;                 - UBUFF will be copied to text register,
;                 - ^Z and all, and will be invisible until
;                 - Mate redraws the screen.
UVAR2:  DW 0
UVAR3:  DW 0
UVAR4:  DW 0
UVAR5:  DW 0
UVAR6:  DW 0
UVAR7:  DW 0
UVAR8:  DW 0
UVAR9:  DW 0

     <part omitted>

        ORG UINIT

        LHLD 06H            ;POINTER TO BEGINNING OF FDOS
     < part omitted>

        in      clkdat  ; Is System Support board
        cpi     0FFh    ; in the system?
        rnz             ; Yes, else
        mvi     a,0C9h  ; Wipe out user calls
        sta     CLK$1   ; numbers 1 and 2.
        RET


        ;*************************************;
        ;*          INITIAL COMMAND          *;
        ;*************************************;

USRCOM:         ; Execute this macro after each BREAK (^C)
      ;db       ´bte xk iThis is my macro$´ (Put yours here)
       db       0       ; End of macro marker.

;*********************************************************
;*                                                       *
;*  Machine language Interface: Gets here via XM command. *
;*                                                       *
;*********************************************************

FERJOY: ; Entered via Mate "XM" command
        push    h
        push    d
        push    b

        lhld    UVAR0               ; Mate stores request
        mov     a,h                 ; # in user variable 0.
        cpi     0                   ; Allow 255 calls max.
        mov     a,l                 ; Param. in A.
        jnz     FER$AL              ; *** Error if not 0 ***
        cpi     FERNUM              ; defined in table?
        lxi     h,FER$AL            ; simulate call instruction
        push    h                   ;
        rnc                         ; NO, not in table, <cr>, else
        mvi     d,0                 ; Calculate routine address
        mov     e,a                 ;
        lxi     h,FERTAB            ;
        dad     d                   ;
        dad     d                   ;´HL now points to vector
        mov     e,m                 ;
        inx     h                   ;
```

```
        mov     d,m                 ;
        xchg                        ;
        pchl                        ; Do it!

FER$AL: pop     b                   ; Do next
        pop     d                   ; stage of
        pop     h                   ; parameter
        ret                         ; test, and back
                                    ; to Mate.
FERTAB: dw      FER$0               ; Jump to monitor
        dw      FER$1               ; Get time
        dw      FER$2               ; Get date
        dw      FER$3               ; Jump to specified address
        dw      FER$4               ; Copy string to here
        dw      FER$5               ; Copy string from here
        dw      FER$6               ; Save current user #
        dw      FER$7               ; Set user #
;       add´em as you see fit.

FERNUM  equ     ($-FERTAB)/2

;*      Get to monitor via ´0q0 xm´ instead of ´xm´.          *
;
FER$0:  jmp     MONITR

;*      Read time/date, store time at (UVAR1)                 *
;
FER$1:  call    CLK$1               ; Time request
        lxi     d,TIMTAB            ; Time text
FER$1A: lhld    UVAR1               ; goes here.
        mvi     b,8                 ; Move this number of bytes.
        jmp     UMOVIT              ; Do it.


;*      Get time/date, store date at (UVAR1)                  *
;
FER$2:  call    CLK$1               ; Date request
        lxi     d,DATTAB            ; Date from here...
        jmp     FER$1A              ; Do it.


UMOVIT: ldax    d                   ; Get byte
        mov     m,a                 ; Store it
        inx     h                   ;
        inx     d                   ;
        dcr     b                   ; More?
        jnz     UMOVIT              ;  yes
        ret                         ; Done

; ***** Next instruction is patched to ´RET´ *****
; *****  if System Support board is missing. *****
CLK$1:  lxi     h,OKITAB            ; Register access table
        lxi     d,PMATAB            ; Store here for Mate
CLK$2:  mov     a,m                 ; What ho!
        cpi     ´$´                 ; Time to go?
        jz      CLK$5               ; Done
        cpi     14                  ; All reg´s. < 14
        jnc     CLK$4               ; Print a seperator
        ori     READHO              ;
        out     CLKCMD              ; Register request
        cpi     READHO + 5          ;
        in      CLKDAT              ; Get data
        jnz     CLK$3               ;
        sui     08h                 ;
CLK$3:  adi     30h                 ; Binary to ASCII
CLK$4:  stax    d                   ; Store in PMATAB
        inx     h                   ;
        inx     d                   ;
        jmp     CLK$2               ;
CLK$5:  stax    d                   ; Loop exit
        xra     a                   ; Take clock
        out     CLKCMD              ; off hold.
        ret                         ;
```



```
;*********************************************************
;*  The following NUMBERS in OKITAB represent the order in  *
;*    which the OKI data registers will be accessed.        *
;*  The time/date will be written to PMATAB, and eventually *
;*    will be copied to a Mate text buffer.  As Mate is not *
;*    aware of this, the time/date will be invisible until  *
;*    Mate is forced to redraw that part of the display.    *
;*********************************************************
```

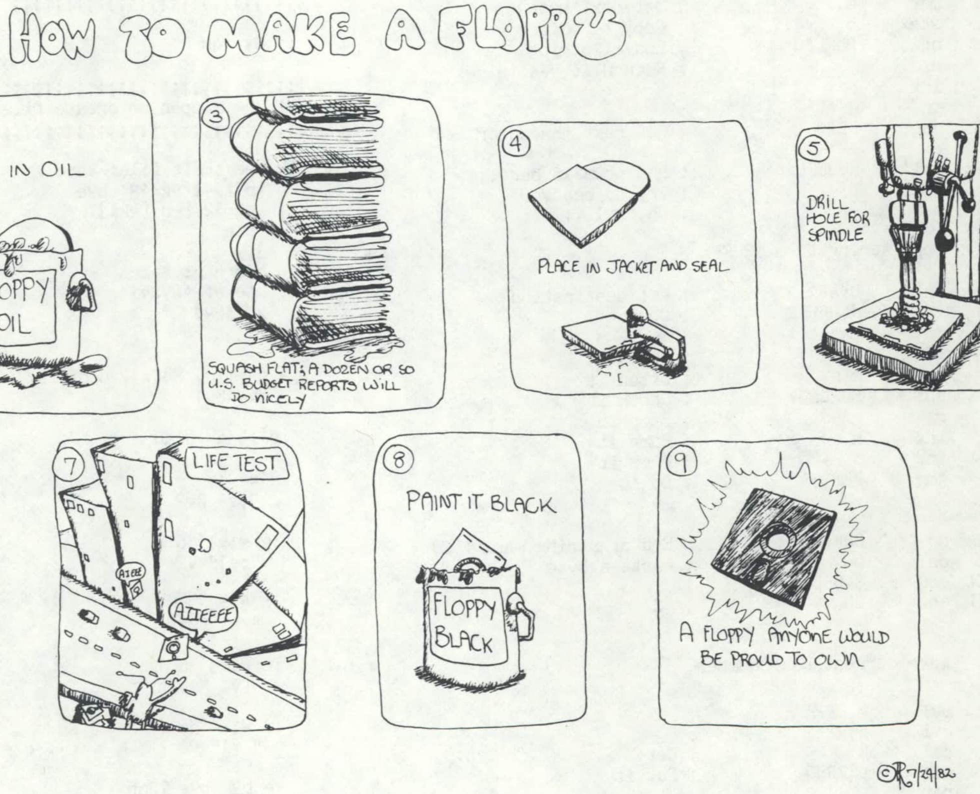
```
OKITAB: db       5,  4,  ´:´,  3,  2,  ´:´,  1,  0
        db      10,  9,  ´/´,  8,  7,  ´/´, 12, 11, ´$´
PMATAB:
TIMTAB: db      ´hh:mm:ss´
DATTAB: db      ´mm/dd/yy$´

;*      Jump to address via ´####q1 3q0 xm´                  *
;*      #### is the address; don´t forget about your radix.  *
;*      **** Haven´t tested for addresses above 32k. ****    *
        ;
FER$3:  lhld    UVAR1               ; Go to this address
        pchl                        ;


;*      Copy string to UBUFF                                 *
;
FER$4:  lhld    UVAR1               ; Points to source string
        lxi     d,UBUFF             ; Destination
        mvi     b,BUFLEN            ; Set limit on max # of
        ;
FER$4A: dcr     b                   ; bytes to move
        jz      FER$4B              ;
        mov     a,m                 ; Get next byte
        stax    d                   ; Copy to buffer
        cpi     BUFLAG              ; End of string?
        rz                          ; Return if yes, else
        inx     h                   ;
        inx     d                   ;
        jmp     FER$4A              ; Get next character
        ;
FER$4B: mvi     a,BUFLAG            ; Bad news if here...
        stax    d                   ; Try to recover
        ret                         ; Oh, well...

;*      Copy string from UBUFF                               *
;
FER$5:  lhld    UVAR1               ; Get destination
        lxi     d,UBUFF             ; Source
        ;
FER$5A: ldax    d                   ; Fetch it
        mov     m,a                 ; Ditch it
        cpi     BUFLAG              ; Check it
        rz                          ; Stop it
        inx     h                   ; Bump it
        inx     d                   ; Bump it
        jmp     FER$5A              ; Do it


UBUFF:  db      BUFLAG              ; End of string mark (^Z)
        ds      80                  ; Waste a byte
BUFLEN  equ     $-UBUFF             ;
BUFLAG  equ     26                  ; ^Z


;*      Save current USER number.                            *
;
FER$6:  mvi     e,0FFh              ;
        mvi     c,32                ;
        call    5                   ; Get #
        sta     USERHO              ; Put it
        ret

;*      Change USER number to new or previous value.         *
;
FER$7:  lhld    UVAR1               ;
        mov     a,l                 ;
        cpi     0FFh                ;
        cz      FER$7B              ; Wants to restore old user #
        mov     e,a                 ;
        mvi     c,32                ; Set new number
        jmp     5                   ; Do it
FER$7B: lda     USERHO              ; Old user #
        ret                         ;

USERHO: db      0

KEYTB:
        ;-- Move to top <-----------------------.
        db 128, ´A´-40h, 0, 0;                  |

   <remainder omitted>


        db 0FFH         ;END OF TABLE


;END OF EDITOR
EDEND:  db 0

        END
```

```
;*****************************************************************
;*    Version: 003,    Time: 23:08:23,    Date: 09/17/82.        *
;*****************************************************************
;*    Program: Permmac                     Author: Bill Norris   *
;*    Permanent macros for PMATE.                                *
;*    Text Registers 8 and 9 reserved for internal use.          *
;*    Also uses: reg. 7, var. 0,1,2 and user var. 0,1.           *
;*                                                               *
;*    ^F^C updates time/date.                                    *
;*    ^F^F edits file, creating/updateing box.                   *
;*    ^F^H displays this box. (PMAC.HLP on current drive)        *
;*    ^F^V updates version #, time/date.                         *
;*    note:                                                      *
;*    ^F^F or .f must be used first (on real or dummy file)      *
;*    in order to be able use ^F^A and ^F^B.                     *
;*    Version, time and date to be updated is first found.       *
;*                                                               *
;*****************************************************************

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
^Xc     ; update time, redraw screen
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 .s 01  qr


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
^Xf     ; open or create file with version/time/date header
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

b9k b9e iEdit file: $bte          ; prompt for filename
   [g^A@9$ @k=13_ b9e             ;
    @k=127[-d][@ki]               ; get filename
    bte
   ]
   b9e a s: $-k                   ; strip prompt
   bte @f^A@9$v1                  ; is file new?
   xf^A@9$                        ; open file
   @1jj                           ; skip prompt if file exists

 :a gAsm=0, M80=1, C or PL/I=2, Pascal=3, Other=4. - Select:  $
   @k-48v1

  @1>4ja  @1<0ja                  ; loop if out of range

  @1=0´j@                         ; if not asm, jmp to @
    ";v1 0v2                      ;  else comment
                                  ;  - characters = ";",,
  :@ @1=1´jb                      ; if not M80, jmp to b
    ";v1" V2                      ;  else comment
                                  ;  - characters = ";"," "
  :b @1=2´jc                      ; if not C or PL/I, jmp to c
    "/v1 "/v2                     ;  else comment
                                  ;  - characters = "/","/"
  :c @1=3´jd                      ; if not Pascal, jmp to d
    "(v1 ")v2                     ;  else comment
                                  ;  - characters = "(",")"
  :d @1=4´je                      ; if not Other, jmp to e
    " v1" v2                      ;  else comment
                                  ; - characters = " "," "
  :e b9k b9e 72qh                 ; insert spaces for header
   8m @1r 62[r*$]                 ;  first comment character, row
       @2r 13i                    ;  second comment character
   b8k b8e 8qh                    ; insert space for time
   z @1i i*$                      ; comment out
   60qh z i*$                     ; body of header
   @2i 13i bte                    ; finish
   .z .v                          ; insert vers/time/date/author
   @1=32´jf                       ; if comment char = " ", jmp to f
   a3l 14m 2n jj                  ; go to overwrite, jmp to j
  :f 2l 14m rProgram:$            ; replace spaces
    23m rAuthor: auth. name $     ; put your name here
   0l s: $ r^A@7$ l 14m 2n        ; position cursor, go to
                                  ; - overwrite mode
  :j b9c -l b8c b8e as*$60d       ; take out old time/date
   60qh bte .v a0g$               ; replace with new

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
^Xs             ; Insert time and date
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

0l stime: $ .t sdate: $ .u


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
^Xt             ; call #1 -- time
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

@Pv9 @9q1                   ; give assembly-language routine
                            ;  address of time string
   1q0 xm                   ;  set call # to 1, do call
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

(continued next page)

```
^Xu             ; call #2 -- date
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

@Pv9 @9q1                  ; give assembly-language routine
                           ;  address of time string
   2q0 xm                  ;  set call # to 2, do call

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
^Xv             ; increment version number
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

0ls: $ 3m 1v0              ; find place
3[.x] 0l.c                 ; increment with carry

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
^Xx             ; increment number under cursor with carry
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

-m @t-48+@0v0 @0<10jk   ; increment number under cursor
```

```
   "0v1 1v0 jl
   :k @0+48v1 0v0
   :l @1r -m

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
^Xz                ; put labels into box
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

b9g b8g b9g 4[b8g] b9g 13i -91  ; insert lines
   s*    $                      ; find place
   rVersion: 000,   Time:           ,    Date:         .$
   @2=0[0l d rsay hey$                  ; replace spaces
       2[s,$d] s.$k                     ; force listing for ASM files
       13ild                            ; clean up
     ]
   -2l.s                                ; insert time/date
^X
```


HOW TO MAKE A FLOPPY
1
CUT A CIRCLE OUT OF CARDBOARD
2
DIP IN OIL
FLOPPY OIL
3
SQUASH FLAT; A DOZEN OR SO U.S. BUDGET REPORTS WILL DO NICELY
4
PLACE IN JACKET AND SEAL
5
DRILL HOLE FOR SPINDLE
6
CUT SKIRT FOR SPINDLE AND OPENING FOR READ/WRITE HEAD, BOTH SIDES
7
LIFE TEST
AIEE
AIIEEEE
8
PAINT IT BLACK
FLOPPY BLACK
9
A FLOPPY ANYONE WOULD BE PROUD TO OWN
©R 7/29/82

iRMX Users Group

Anne Odden

The National iRMX Users Group has been formed, sponsored by Intel and Lifeboat Associates, who have helped the group establish a solid foundation of interest and financial backing.

One objective of The National iRMX Users Group is to publish a newsletter which will provide useful information to the iRMX system user. This publication is called *Human Interface* and will appear quarterly. The iRMX Users Group has also established its software library in New York City. The library will offer members and iRMX system users various volumes for a nominal fee. The library will begin taking orders in late November.

Membership charges in iRUG (iRMX Users Group) may be either a corporate fee of $50.00 or an individual fee of $10.00. All members receive the newsletter, which is available only to members. Library diskettes are available to members and the public for $15.00. Memberships must be renewed annually. (The corporate membership includes a discount on software.) All orders must be prepaid by check, VISA, MasterCard or American Express.

Since the introduction of the iRMX operating system, many local users groups have formed. Two such groups are located in Texas and Wisconsin. Those interested in forming a local N.Y. Metropolitan area iRMX Users Group, please call Lifeboat Associates at (212) 860-0300, ext. 350.

By November iRUG expects to have complete information available; if you wish to receive material describing iRUG and its services, please call or send your address to The iRMX Users Group, 1651 Third Avenue, New York, N.Y. 10028.

Software Notes
Improvements In Pascal/MT+ Version 5.5

Reported by Al Bloch

This full ISO standard implementation of the popular Pascal programming language is loaded with additional features, such as very friendly editing and pre-compile error-checking environment, along with practical extensions, including in-line assembly code, modular compilation, overlays, bit and byte manipulation, and string and file handling commands. Since its release in late 1981, user experience has elucidated several corrections and patches to the distributed code; while not warranting a new version release (which would require updating), they add considerably to convenience and accuracy of use.

First, in implementing the Speed Programming package (SPP – the editor, syntax checker and precompiler), the user must insert into the file NSB.SRC appropriate Pascal code to generate the cursor-handling commands required by his or her CRT. NSB.SRC calls as an INCLUDE file another source listing on the distribution diskette, EDTYPES.SRC, which contains standardized CONSTANT and TYPE declarations, quite a few of which are not required for SPP use. The impact of this is minimal on 64K machines, but prohibitive in a 56K environment, where the unnecessary swelling of the symbol table renders NSB uncompilable.

A simple correction is to comment out the unused constants in EDTYPES.SRC before compiling NSB, as follows (parenthetical comments are mine):

```
CONST
     (*XINSRT=0
         ... (another 32 lines)
         CHINS=$1A; {Z} *)
         ESC  =$1B; (leave this in)
     (* QUIT =$FE;...          *)
         CMDSZ=23;  (leave this in)
     (* LF   =$0A;
         ... (another 6 lines)
         FIRSTLINE=0 *)
         STRLEN   =30;
         (this one, and
          the following,
          are all required)
```

Next, a series of patches have been defined to correct various discrepancies and system dependences, of various sizes and complexities. The details are available from Digital Research and their dealers; the list in Figure 1 outlines what they're good for.

Miscellaneous new lessons:

1- PLINK II users can link Pascal MT+ .ERL files, by adding the following code to the MAIN module:

```
CONST  @xxxx1= $103;
       @xxxx2 = 0;
```

(A similar patch for Pascal/Z was noted in *Lifelines*, July 1982.) These two variables are initialized ordinarily *only* by MT+'s own overlay handler.

2- Documentation typos:
pg. 68 Demo prog. IF line contains a proper ')' which is missing from the file on disk;
pg. 211 with AMD 9511, one must also still link FPREALS/S before PASLIB. AMDIO, FPRTNS, REALIO and TRANS9511 must be on the logged-in drive.

SPP Manual Addendum, pp. 4 & 5. [addr (sb__out__ch] at bottom and top lines need a ')' before ']' twice. User must also delete (*$K7*) at head of NSB.SRC to implement this patch.

3- If your LINKMT dies on a HP-125 after a seemingly successful link, just when it's ready to write the output to disk, talk to HP about an updated version of BIOS for their CP/M; early 1.1's had this problem.

4- Assembly code .REL files produced by RMAC or MACRO-80 may be linked into MT+ program by simply RENaming them to .ERL.

Figure 1

PATCH #	PROBLEM SOLVED	METHOD
01	PASTEMP.TOK under 1.4 CP/M	ddt MTPLUS.000, 1 byte
02	File Name Syntax under 1.4 CP/M	ddt LINKMT.COM, 1 byte
03	Error #4 in INLINE assembly using JMP(*+n)or(*-n)	ddt MTPLUS.COM, 3 instrux
04	EXP(-n) shouldn't equal EXP(n)	ddt TRANCEND.ERL, 17 bytes
05	Source in FIBDEF.LIB,RNB.SRC and ATWNB.SRC does not compile to match code in PASLIB.ERL	4 pages of new source code
06	Formatted string output off by one in size (too short)	16 line new source code
07	@OVS in PASLIB.ERL can't find overlays on drives other than logged-in one.	ddt PASLIB.ERL, 2 bytes
08	SPP DIR function under 1.4 CP/M	2 pages of new source code
09	/X switch for overlays didn't work	ddt LINKMT.COM, 1 byte
10	"invalid opcode" in compile phase 2, with deeply nested IF-THEN-ELSE or large CASE statement; hardware stack overflow.	ddt MTPLUS.000, 1 instrux

Notes: CDOS, MUON and CP/Ms on Cromemco hardware are treated as 1.4 CP/M. Patch 6 does *not* resolve surprises in printing floating point format; this one is still under study by the authors.

Software Notes

PANEL Overview

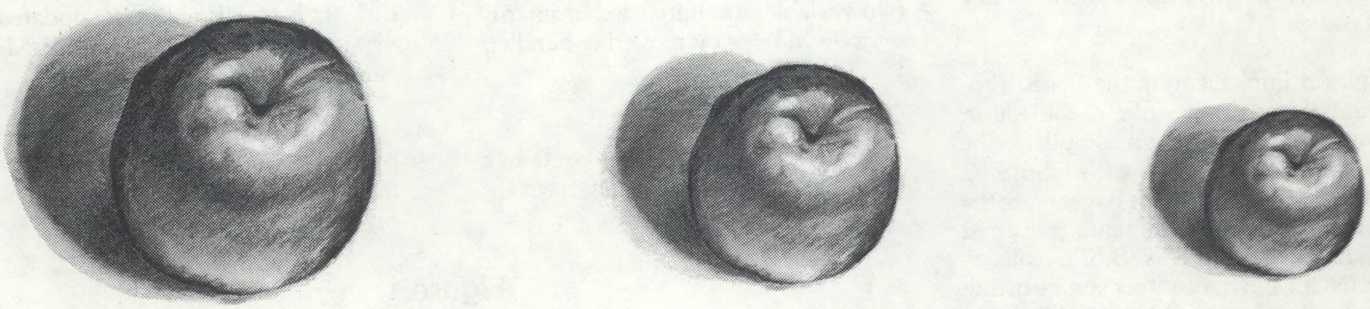
Jethro Wright III

In an upcoming issue of *Lifelines,* we will present an in-depth review of the PANEL data entry design system, as part of our continuing series of Applications Development Software. PANEL comes from Roundhill Computer Systems Limited, distributed exclusively by Lifeboat Associates.

As mentioned above, PANEL is a system of interative programs that facilitates the transfer of information to and from a CRT terminal and/or a conventional data base. However, unlike similar tools in the marketplace today (including those programs previously reviewed in *Lifelines*), PANEL is not "wired into" a single language — proprietary or otherwise — but can be used with COBOL-80, Pascal/MT+, or PL/I-80. The programmer/designer interfaces his/her program to a panel (screen layout) via a series of pre-defined subroutines that perform almost every desired function in a conventional transaction. In addition, PANEL comes with alternative programs that take the place of user-written applications programs when the requirements of these tasks are less sophisticated. These programs can capture data from the screen as well as manipulate large collections of data already stored in a file.

Not only will we be offering a penetrating review of this exciting product, but we will also have examples of what it takes to make it all work, in the form of a simple application program.

Watch for it.



Attention Dealers!

There are a lot of reasons why you should be carrying Lifelines/The Software Magazine in your store. To provide the fullest possible service to your customers, you must make this unique publication available. It will keep them up to date on the changing world of software: on updates, new products, and techniques that will help them use the packages you sell. Lifelines can back up the guidance you give your customers, with solid facts on the capabilities of different products and their suitability to a variety of situations. Now we can also offer you an index of all back issues of Lifelines, opening up a full library of information for you and your customers.

For information on our dealer package, call (212) 722-1700, or write to Lifelines Dealer Dept., 1651 Third Ave., New York, N.Y. 10028.

Product Status Reports

The products described below are available from their authors, computer stores, software publishers and distributors. Information has been derived from material supplied by the authors or their agents, and *Lifelines/The Software Magazine* can assume no responsibility for its veracity. Software of interest to our readers will be tested and reviewed in depth at a later date.

New Products

COMPress

Digital Marketing
This program may reduce archival storage requirements by 30-40%, using a technique of byte-to-byte encoding. Shorter bit codes are assigned to more frequent characters and longer codes to less frequent ones. Data can be compressed without loss of data, allowing faster transmission for communications. COMPress works with ASCII and non-ASCII files. The program requires CP/M-80. Price: $59.95

Expense Track

Sapana Micro Software
This program keeps track of expenses throughout the year and at tax filing time; it is appropriate for home and small business use. Seven fields for data entry are provided: date, description, category (1-99), method of payment, tax status, and amount. Almost 2500 expense entries can be stored on a single-sided floppy. Entries can be deleted or modified. 64K, an 80-column display, one drive, PC-DOS and a printer are required. Price: $29

4th

United Controls Corporation
This compact interactive package is intended to provide a full software development environment. It includes a command line interpreter with a calculator mode, assembly and compilation features, program execution and debug, and nested source file loading.

The 4th language is designed to produce full structured code and data; it is highly modular and claims to encourage top-down design with bottom up coding. Single and double precision integers, booleans, strings, arrays and records are supported.

The assembler features 8080 mnemonics. Also included with the package are an editor, a tracer/debugger and a cross-compiler. The debugger permits run-time stack display and interactive patching of previously compiled code. It can decompile and disassemble all 4th language code. The cross-compiler allows generation of ROMable code and produces compact COM files.

48K RAM and CP/M-80 are required for this product. Price: $89.95

LAZYCODER SCREEN

Nelson Data Resources, Inc.
This presentation development aid allows the design of images or data entry screens and includes thirty-five design functions. A screen can be printed, saved for data entry, incorporated into a series of such displays, or filed with a filing system option. Price: $125

Mail Track I

Sapana Micro Software
This mail processing package stores 110 labels on a single-sided floppy and 2200 on a double-sided disk. The mail list remains in Zip code order as information is entered, and duplicate entries are flagged. Searches can be executed using any of the seven fields provided. Foreign entries are supported.

Labels can be printed one, two, three or four across. Entries can be moved from one file to another, edited.

64K, one drive, PC-DOS and a printer are required. Price: $29

Manufacturing Control System

Microcomputer Consultants
Consisting of several smaller independent packages, this product is designed to address individual manufacturers' particular needs. The packages form a Manufacturing Resources Planning System, encompassing manufacturing controls from sales department forecasts through production.

The inventory control module performs standard functions and also provides bill of materials processing and job tracking. Product structures for assembled items are maintained and gross material requirements reports produced; part usage is tracked. Four costing methods are supported.

Sales order entry and purchase order entry modules tie in with accounts receivable and accounts payable.

Written in PL/I-80 and utilizing Access Manager for file handling, these modules run under CP/M-80 or MP/M-80. Pricing information is not yet available.

POWER

COMPUTING!
This multi-utility program is a menu-controlled interface for CP/M-80. Files are listed on a screen menu and manipulated using numbers which POWER assigns to them. A reclaim function allows the user to restore accidentally erased files; a separate disk test feature permits bad sectors to be gathered into a special invisible file.

Fifty housekeeping programs are included in the package, which occupies 12K. Monitor commands read and write to any selected track or sector from any location in memory. POWER permits a user to fill memory, move memory and single step in any direction, entering ASCII, hexadecimal, decimal or binary code. Memory can be searched using wildcards. Programs can be executed in any memory location.

CP/M is required. Price: $149

Super Generator/Super Indexer

Winsoft
These packages can be used together, or may be purchased and used independently. They are designed for the production and maintenance of professional documentation. The Report Generator adds title page, table of contents, list of figures and tables, revision list, section numbering (to seven

(continued next page)

levels), section relative or global pagination, three kinds of heading, four kinds of margins, visible page borders, lettered appendices, and fifty other features. Some of these include multiple text columnization, text width expansion and contraction, right justification, two types of footnotes, and table heading carryover.

The Super Indexer utilizes words specified in a list or parametrically for indexing. Dehyphenation is automatic and other features include: indexing in exact case or not, indexing of word variants under a single entry, and referencing of one index entry by another.

The system is available for CP/M-80, IBM PC DOS, UCSD Pascal, and other computers. The Indexer and Report Generator are priced at $600, or $1100 for both; manuals cost $25 each, applicable towards the purchase price.

TCW/DMS

The Computer Workshop
This data management system is designed for users with little technical knowledge. Twenty programs and three levels of complexity are included.

The number of possible files is limited only by disk capacity; twenty-four fields are permitted per file. Each field may be alphabetical, numeric, or a date. Length and decimal points may be specified for numeric fields; length may be specified for alphabetic fields. An index key allows quick record location; if a key is specified, a hash code location technique is implemented. Field specifications may be saved for future use.

A field to be computed may be specified. Formulae for computed fields may contain names of variables, constants, *, /, =, + and minus, along with parentheses to indicate precedence. Index keys can be updated automatically. Information from the previous record may be repeated during entry.

Numeric fields are checked for proper content. Add or subtract operations can be used to increase or decrease amounts in numeric fields, and computed fields are automatically updated.

Deleted records remain until a file is compressed, and can be restored. The compress function also automatically makes a backup copy of the data file.

Records can be located by value, position or index key. Up to twenty-four keys can be specified for sorting on, in ascending or descending sequence. Compressed files can be re-sorted.

The user may, at the middle complexity level of TCW/DMS, format reports, citing headings, fields, captions, totals, etc. Files can be converted into standard sequential format with commas between fields, and back to TCW format. At this level a query language can be utilized, as can the X-Y plotting capability of the product. Simple statistics are supported; maximum, minimum, sum, count, average, variance and standard deviation of numeric fields can be computed.

At the third and highest level of complexity, two files can be merged into a third when keys between them match, creating a new file with selected information from both input files. The number length, type name or position of fields can be changed, and fields may be added or deleted.

At this level the user can employ a preprocessor so that BASIC programs can access TCW/DMS files. Symbolic field names in a data file can be treated as variables. Input/output operations are supported to facilitate the interaction with the user's BASIC programs.

File linkage features link records from different files; records are related hierarchically through values in fields. There are no imbedded pointers within the records, and a program is supplied to allow navigation through the linked structure.

TCW/DMS requires 64K of memory, IBM PC DOS, IBM PC BASIC, two drives, a printer, and a monitor.

TE100 Terminal Emulator

Persoft, Inc.
This product emulates most features of DEC VT52, VT100, VT101, or VT102 terminals, allowing an IBM PC to function as these terminals do. Setup mode, character attributes, line and character insert and delete, and modification of terminal characteristics from the host system are supported.

The baud rate ranges from 75 to 9600; the screen format supports 24 by 80 lines, with a twenty-fifth line for status and indicator light display. US or European ASCII character sets and line drawing graphics are featured, along with split and reverse screen functions, bold, blinking and underline. The numeric/function pad is controlled by the host computer.

Local or remote applications (via modem) are supported, along with a local echo option and local printer support.

An IBM PC with one disk drive, an asynchronous I/O board with cables, MS DOS and 64K of memory are required. Price: $125

Z80 Relocating Macroassembler

2500 AD Software, Inc.
This package includes a linker which will link over four hundred files, along with an 8080 to Z80 Source Code Converter. Files can be as large as the user's disk storage space, because buffers may overflow to the disk. The program also assembles files with nested macros to an unlimited number of levels.

Command line or self prompted invocation is supported as are all Zilog mnemonics, syntax and directives. Listing options include Pass One only, terminal only, or printer only; sections of code can be listed during assembly.
Price: $49.50

New Versions

This month the following products have been updated to the version numbers noted. We hope to have more information on these updates next month.

ASCOM/86	2.10
FABS-II	4.17
MATH*	3.044
MATH-PC	3.0
T.I.M. III PC	3.20

The new version of T.I.M. is compiled. See Macros of the Month for news on PMATE updates. This issue also includes a Software Note on the latest version of Pascal MT/+. Below is described a new version about which we have received detailed information.

Lattice 8086/8088 C Compiler

Version 1.01
The source for the basic console I/O function has been supplied, and may be customized by the user to suit individual needs. In addition the source for the function extract utility is included, along with three new macros.

A special compile time option has been added to the compiler's first phase; this option allows a drive other than A: to be specified when a program is compiled. The "kbhit" function has also been added. It returns zero if a character has not been typed at the keyboard and non-zero if a character is pending; this action is opposite to that described in the manual.

Initializer expressions for declarations forced to "extern" status by the -x option are now ignored, instead of being flagged as errors. If the first phase of the compile processes an input file which doesn't declare any functions or data, a message will be generated and execution terminated; the .Q file will be deleted.

The library routines which process the read and write functions for disk files now obtain the disk blocking factor from the the external location, instead of always using a default value of 128. Block values of 256, 512, and 1024 now will also be supported, and the block value should be defined by initialization.

The "rstmem" function has been changed so that only allocations made after a call to "allmem" are affected, letting programmers make a certain number of initial "sbrk" or "getmem" calls and then initialize a memory pool by calling "allmem". The restriction that "rstmem" cannot be called if any files are open no longer applies unless the files were opened after the first call to "allmem".

The "clrerr" function now clears the end of file flag.

Programmers may now create .EXE files which can be converted to .COM files using the EXE2BIN utility implemented in recent versions of MS DOS. The program must be linked according to directions supplied in a manual addendum incorporating this update.

Books

Reviewed by Raymond Sonoff

BASIC for Business
Douglas Hergert
SYBEX, Inc. 1982
Berkeley, California

This is a most PRACTICAL book on learning BASIC language. Mr. Hergert has created a model of exposition for others to study and to apply. Principles and concepts of the BASIC language are presented with such a smooth coordination that the usual difficulties – first trying to learn definitions and new terminology, then trying to appreciate the significance of concepts, attempting to translate all such new material into actual business applications programs that work, endeavoring to achieve a user-friendly, interactive, well-formatted program, etc. – are avoided.

That the author believes in and practices taking a high road of understanding rather than adopting the often-followed hacker's approach is supported by the lack of programming shortcuts. Elegance of solution shines through in the end, however. The author concentrates on providing supporting material that illustrates fundamentals and interrelationships among BASIC's reserved words, control structures, program structure, formatting, etc.

The complete BASIC business applications programs cited in each chapter do testify to the fact that the approach used does result in direct solutions to problems of the everyday business world. Similarities and differences between BASIC and other popular languages, such as COBOL, Pascal, and FORTRAN, are included in each of the seven chapters of the 223-page 7″ x 9″ softcover book. In fact, Appendix B gives complete programs in each of these languages so that you can directly compare a given program with the corresponding one already presented using the BASIC language.

Among the numerous programs presented in the book are Cost of Goods Sold, Comparative Income Statement, Discount Factors, Improved Annuity, Depreciation, Present Value of a Depreciation, Cost-Volume-Profit Analysis, and Break-Even Point.

Written for the business professional, this book should aid any reader to read, write, and to "debug" BASIC programs. And, after all, learning by doing is the principal way to find out what a computer (and you) can do.

Something New

When you can't find your problem, let ACTIVE TRACE show it to you! See inside your program as it's working! Just as important, see inside your program when it's *not quite* working!

New to Basic? ACTIVE TRACE will let you see what Basic does as it does it! ACTIVE TRACE displays the line number, name, and current value of the variables and functions you choose, as they are encountered in program flow.

Something Old

Though less exciting than harnessing the power and speed of your computer to find mistakes, using your computer to avoid mistakes in the first place is equally valuable. Cross-reference utilities have been around for a long time. Most programmers would not attempt to work without them, and we don't know why they have not become more well known and understood among Basic programmers and educators. ACTIVE TRACE produces complete cross-reference maps and *explains their use and importance.*

Active Trace

If you have great intuition and are well-disciplined, then you'll want ACTIVE TRACE. But if you're like the rest of us, you need ACTIVE TRACE to:

- Understand and modify programs you did not write
- improve your programming skills
- minimize program development time

$125.00

complete with primer to help you use ACTIVE TRACE to improve your programming.

Why pay more for cross-reference utilities alone when you can have ACTIVE TRACE, the new easy to use programming environment for the Microsoft family of Basic interpreters.

SOFTWARE
SOFTWARE
DIGITAL MARKETING
DIGITAL MARKETING™

DIGITAL MARKETING CORPORATION

2670 CHERRY LANE • WALNUT CREEK • CALIFORNIA • 94596
(415) 938-2880 • Telex 17-1852 (DIGMKTG WNCK)

ACTIVE TRACE is a Trademark of The Data Works.

(continued next page)

Software Notes

Tips & Techniques

Wells Brimhall of Phoenix, AZ has sent in this routine.

"Anyone who has used an ADM-3A CRT with the lower case option has most likely cursed Lear Siegler for not including a shift lock key. There is a lower case disable switch on the main PC board, but it is a real pain to switch it back and forth when editing a file. After spending some time trying to convert one of the less used keys to a shift lock through hardware, I realized that it can be done very simply with software. This short routine must be installed in your CP/M BIOS CRT driver. It translates the key of your choice to upper case lock.

With this routine the CRT will always boot up in the upper case mode, and all alpha characters will be converted to upper case. When you press the shift lock key the keyboard will return to the normal upper/lower case mode. Each time the shift lock key is pressed the keyboard will toggle back and forth between upper case only and upper/lower case. To enter the ASCII code for the key that shift lock replaced, simply press the key twice. After a few tries I decided that the backslash key works best. It is located in the right place and is seldom used. A control character could also be used, but two fingers can be quite inconvenient. This routine also swaps rubout and underline so you don't have to use the shift key to rub out mistakes.

To implement this routine in your BIOS you must:

1- Add the routine.

2- Change your present CRTIN entry point to route CRT input calls through ADMIN instead of your old CRT input driver.

3- Change the two CRTIN calls in this routine to call whatever your old CRT input driver is labeled. Your old driver is still used without modification.

I have only done this on my ADM-3A CRT, but the technique should work for any type of terminal that needs a shift lock key."

Input Driver

```
0=zero  O=letter O
; + + +  ADMIN  + + +

;Input driver for LEAR SIEGLER ADM-3A CRT

;A shift lock character is supported (SLOCK).  All lower case
;letters typed after a shift lock will be converted to upper
;case until the next shift lock is typed. Typing 2 successive
;shift locks will cause the shift lock character to be input.

;UNDERLINE and RUBOUT are swapped so it is not necessary to
;use the shift key for RUBOUT.

        EXTRN CRTIN

TRUE    EQU 0FFH
SLOCK   EQU 1CH      ;shift lock character, ´\´
RUBOUT  EQU 7FH      ;ASCII RUBOUT
ULINE   EQU 5FH      ;ASCII UNDERLINE

ADMIN:  CALL CRTIN        ;get character from regular routine

;Process shift lock character

        MOV C,A      ;save input character in C
        CPI SLOCK
        JNZ ADMIN2   ;JMP if not shift lock

;Shift lock was input, if the next character is
;another shift lock then return it to the program,
;otherwise toggle the upper case flag byte.

ADMIN1: CALL CRTIN   ;get another character
        CPI SLOCK    ;2nd shift lock?
        RZ           ;RET if 2 successive shift locks
        MOV C,A      ;save 2nd character in C
        LDA UCASE
        CMA
        STA UCASE    ;toggle upper case flag

;Convert character to upper case if UCASE is TRUE

ADMIN2: LDA UCASE
        ORA A
        MOV A,C      ;A=input character
        JZ ADMIN3    ;JMP if UCASE is 0
        CPI 61H      ;61H=lower case A
        JC ADMIN3    ;JMP if character < lower case A
        CPI 7BH      ;7BH= lower case Z + 1
        JNC ADMIN3   ;JMP if character > lower case Z
        ANI 5FH      ;convert to upper case

;Swap RUBOUT and UNDERLINE

ADMIN3: CPI RUBOUT
        JNZ ADMIN4   ;JMP if not RUBOUT
        MVI A,ULINE  ;change RUBOUT TO UNDERLINE
        RET

ADMIN4: CPI ULINE
        RNZ          ;RET if not UNDERLINE
        MVI A,RUBOUT ;change UNDERLINE to RUBOUT
        RET

UCASE:  DB TRUE      ;upper case flag, TRUE=upper case

        END
```

Here's a handy program for use with Pascal MT+, from Dave Miller of Klein Software in Salem, N.H.

The SPP buffer RECOVER program is used when a BDOS or BIOS error occurs while writing the SPP buffer to disk (caused by write protected disk, failure to warm boot when changing disks, or a myriad of other obscure and existential reasons) seemingly destroying the buffer contents. Actually, the buffer contents are still in memory and can be recovered using DDT through a tedious and error prone process. The RECOVER program is designed to simplify recovery.

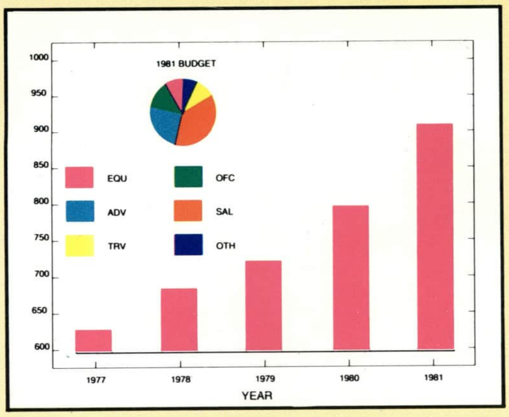
The program should be compiled under MT+ and linked with PASLIB/S to create the RECOVER.COM file. To use: after the system crashes in SPP, insert the disk with RECOVER.COM and execute and warm boot (<ctrl>C). RUN RECOVER and follow the prompts to recover the buffer contents and write them to a temporary file which can then be checked to make sure no damage has been done to the file contents.

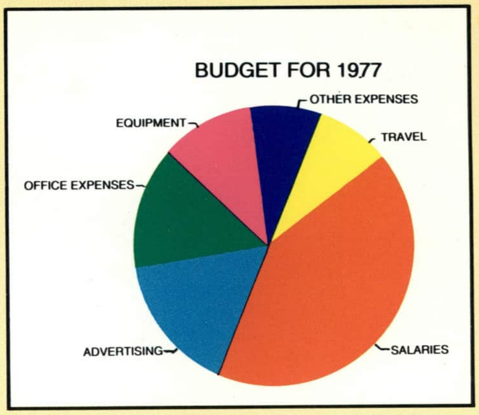
Recover Source Listing

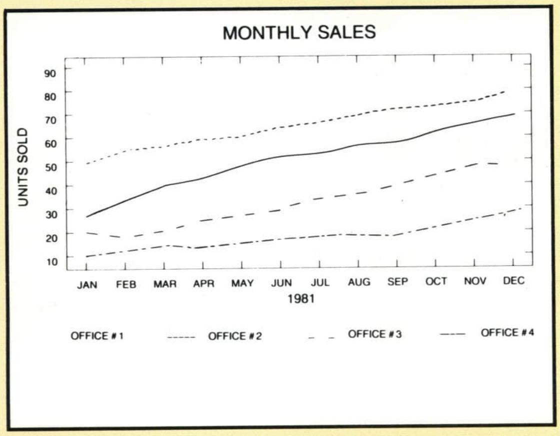
```
TYPE B:RECOVER.SRC

(*$Z $9FFF*)

(* 8/13/1982 D. Miller, Klein Associates, Salem, NH 03079 *)

PROGRAM RecoverSppBuffer;

VAR OUTFILE;FILE OF CHAR;
    TEXTBUFF:ABSOLUTE [$A000] ARRAY[0..0] OF CHAR;
    RESULT,I:INTEGER;
    NAME:STRING;

BEGIN
  WRITELN('This program attempts to save the contents of the SPP');
  WRITELN('buffer when a BDOS error crashes the system.');
  WRITELN;
  WRITELN('The buffer contents start at $A000 and end with a $1A char.')
  WRITELN('    (examinable using DDT)');
  WRITELN;
  WRITELN;('Enter the name of the temporary file you wish to save to:');
  WRITELN('      (form:  [d:] filename.ext)');
  READLN(NAME);
  ASSIGN(OUTFILE,NAME);
  REWRITE(OUTFILE);
  IF IORESULT=225
    THEN
      WRITELN('Error creating ',NAME,' file.')
    ELSE
      BEGIN
        I:=-1;
        REPEAT
          I:=I+1;
          OUTFILE^:=TEXTBUFF[I];
          PUT(OUTFILE);
        UNTIL TEXTBUFF[I]=CHR(26); (*<ctrl> Z EOF CHAR *)
        CLOSE(OUTFILE,RESULT);
        IF RESULT=255
          THEN WRITELN('Error closing ',NAME,' file.')
          ELSE WRITELN('Process completed.');
    END;
END.
```

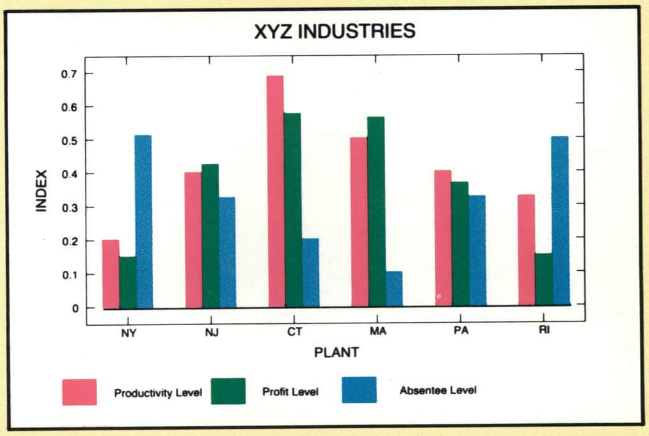
(continued from page 28)

```
        DCR     A       ; convert to pure binary
        CPI     0
        JZ      SETICON ; set console
        CPI     1
        JZ      SETIRDR ; set reader
        CPI     2
        JZ      SETIPUN ; set punch
SETILST:
        STC
        CMC             ; clear carry
        MOV     A,B     ; phys. device
        RAR
        RAR             ; rotate list device into place
        RAR             ; one more to pass carry
        MOV     B,A
        MOV     A,C     ; recover IOBYTE
        ANI     03FH    ; remove old list assignments
        ORA     B       ; combine with new
        JMP     SETI3
SETICON:
        MOV     A,C     ; recover IOBYTE
        ANI     0FCH    ; remove the old console
        ORA     B       ; combine with the new
        JMP     SETI3
SETIRDR:
        STC
        CMC             ; clear carry
        MOV     A,B     ; phys. device
        RAL
        RAL             ; rotate to reader position
        MOV     B,A
        MOV     A,C
        ANI     0F3H    ; remove old reader
        ORA     B       ; combine with new
        JMP     SETI3
SETIPUN:
        STC
        CMC             ; clear carry
        MOV     A,B     ; phys. device
        RAL
        RAL
        RAL
        RAL             ; position at punch field
        MOV     B,A
        MOV     A,C
        ANI     0CFH    ; out with the old,....
        ORA     B       ; in with the new.
SETI3:  MOV     E,A
        MVI     C,SETIOB
        CALL    BDOS    ; set new IOBYTE
        RET


DEFGET: SUI     ´0´ ; make to binary
        DCR     A   ; adjust for range
        RAL         ; times 2
        PUSH    PSW
        MOV     E,A ; first, find proper
        MVI     D,0 ;    logical device name
        LXI     H,LOGDEV
        DAD     D
        MOV     E,M
        INX     H
        MOV     D,M
        CALL    PSTRING ; print it
        POP     PSW
        RAL
        RAL         ; final count times 8
        MOV     E,A
        MVI     D,0
        RET

FIELD:  STC
        CMC         ; clear the carry flag
        MOV     C,A ; save iobyte
        MOV     A,B ; get mask
        MVI     E,0 ; clear counter for rotates
FIELD1: RAR         ; rotate until carry is set,
        INR     E
        JNC     FIELD1
        DCR     E   ; then back up one rotate
        RAL

; field mask now aligned on bits 0 & 1, and E = count of rotates

        MOV     B,A ; put mask back
        MOV     A,C ; recover original field mask
        DCR     E
        INR     E   ; check for already zero
        JZ      FIELD3
FIELD2: RAR
        DCR     E   ; rotate to count in E
        JNZ     FIELD2
FIELD3: ANA     B   ; now isolate bits
        ADD     A   ; double it
        MOV     E,A
        MVI     D,0
        DAD     D   ; finally find index
        MOV     E,M ; get the real address
        INX     H
        MOV     D,M
        RET

; initialize routine.  Print opening remarks and clear
; and /or set various counters & flags.

INIT:   LDA     CBUF    ; if command on command line, user knows
        ORA     A       ; what he is doing, so don´t
        JNZ     INIT1   ; print things he already knows.
        LXI     D,OPNMSG
INIT0:  CALL    PSTRING ; print titles, etc.
        XRA     A
INIT1:  STA     CFLAG   ; set flag for command line input.
        XRA     A
        STA     ALTFLAG
        CALL    CRLF
        LXI     H,OFCBX
        MVI     B,24    ; fill FCBs with zeros
INIT2:  MVI     M,0
        INX     H
        DCR     B
        JNZ     INIT2
        LXI     H,PFCBX
        MVI     B,24
INIT3:  MVI     M,0
        INX     H
        DCR     B
        JNZ     INIT3
        RET

; convert the character in ACC to uppercase if it is
; lower case letter

UCASE:  CPI     ´a´
        RC
        CPI     ´z´+1
        RNC
        ANI     5FH
        RET

; print string pointed to by DE register.  String must be
; terminated by a ´$´, as per CP/M conventions.

PSTRING:
        MVI     C,PRTBUF
        JMP     BDOS

; send a CR, LF to the Console

CRLF:   LXI     D,CRLFMSG
        JMP     PSTRING

        ;data areas

OPNMSG: DB      tab,tab,´IOBYTE Control Utility, Version 1.0´,cr,lf
        db      tab,tab,´   Written by Thomas N. Hill´,cr,lf
        DB      tab,tab,´      June 20, 1982´,cr,lf,lf,lf
MENU:   DB      ´COMMAND MENU:´,cr,lf,lf
        DB      tab,´WHERE      Displays current IOBYTE device assignments´
        DB      cr,lf
        DB      tab,´WHAT       Displays available IOBYTE assignment options´
        DB      cr,lf
        DB      tab,´SET        Allows user to alter IOBYTE settings´
        DB      cr,lf
        DB      tab,´DEFINE     Allows user to define IOBYTE device names´
        DB      cr,lf
        DB      tab,´<CR>       RETURN at prompt returns to CP/M.´
        DB      cr,lf,´$´
PROMPT: DB      CR,LF,´-->$´

SAVEMSG:DB      ´Save new definitions or return to menu (Y/N/R)?´
CRLFMSG:DB      CR,LF,´$´
CONSOLE:DB      ´Console is currently assigned to --> $´
READER: DB      ´Reader is currently assigned to ---> $´
PUNCH:  DB      ´Punch is currently assigned to ----> $´
LIST:   DB      ´List is currently assigned to -----> $´
CONMSG1:DB      ´CONSOLE may be assigned to;´,cr,lf,´$´
RDRMSG1:DB      ´READER may be assigned to;´,cr,lf,´$´
PNCMSG1:DB      ´PUNCH may be assigned to;´,cr,lf,´$´
LSTMSG1:DB      ´LIST may be assigned to:´,cr,lf,´$´
DEFWHAT:DB      ´Enter number of logical device:´,cr,lf
        DB      tab,´1. CONSOLE´,cr,lf
        DB      tab,´2. READER´,cr,lf
        DB      tab,´3. PUNCH´,cr,lf
        DB      tab,´4. LIST´,cr,lf,lf,´$´
CURMSG: DB      ´   Current assignments are:´,cr,lf,´$´
CHANGE: DB      ´        Change to -> ´,´$´
TOOBIG: DB      bell,´Name as entered is too long, make < 24 chars.´
        DB      cr,lf,´$´
DSKERR0:DB      BELL,´OPEN OR CLOSE ERROR DURING PROGRAM UPDATING.´,CR,LF,´$´
DSKERR1:DB      BELL,´DISK WRITE ERROR DURING PROGRAM UPDATE.´,CR,LF,´$´
SELASK: DB      ´Enter the number of the new I/O device: ´,´$´
CERMSG: DB      bell,´Invalid Command, please re-enter from the Menu.´
        DB      cr,lf,´$´

; iobyte field names

; lookup table of string addresses

CNAMES: DW      CTTY,CCRT,CBAT,CUC1
RNAMES: DW      RTTY,RPTR,RUR1,RUR2
PNAMES: DW      PTTY,PPTP,PUP1,PUP2
LNAMES: DW      LTTY,LCRT,LLPT,LUL1

LOGDEV: DW      CONS
        DW      READ
        DW      PUN
        DW      LST

CONS:   DB      ´CONSOLE$´
READ:   DB      ´READER$´
PUN:    DB      ´PUNCH$´
LST:    DB      ´LIST$´

; name strings here.
; since user may assign new names, allow
; up to 24 chars per name.
; CONSOLE field names

CTTY:   DB      ´TTY:$´
        DS      20
CCRT:   DB      ´CRT:$´
        DS      20
CBAT:   DB      ´BAT:$´
        DS      20
CUC1:   DB      ´UC1:$´
        DS      20

; READER field names

RTTY:   DB      ´TTY:$´
        DS      20
RPTR:   DB      ´PTR:$´
        DS      20
RUR1:   DB      ´UR1:$´
        DS      20
RUR2:   DB      ´UR2:$´
        DS      20

; PUNCH field names

PTTY:   DB      ´TTY:$´
        DS      20
PPTP:   DB      ´PTP:$´
        DS      20
PUP1:   DB      ´UP1:$´
        DS      20
PUP2:   DB      ´UP2:$´
        DS      20

; LIST field names

LTTY:   DB      ´TTY:$´
        DS      20
LCRT:   DB      ´CRT:$´
        DS      20
LLPT:   DB      ´LPT:$´
        DS      20
LUL1:   DB      ´UL1:$´
        DS      20

; command table and addresses

CTABLE: DB      ´WHERE´,0
        DW      WHERE
        DB      ´WHAT´,0
        DW      WHAT
        DB      ´SET´,0
        DW      SETIBYTE
        DB      ´DEFINE´,0
        DW      DEFINE
        DB      0FFH

; file control blocks

OLDFCB: DB      0,´SETIO   COM´
OFCBX:  DW      0,0,0,0,0,0,0,0,0,0,0,0
PFCB:   DB      0,´SETIO   $$$´
PFCBX:  DW      0,0,0,0,0,0,0,0,0,0,0,0
RENFCB: DB      0,´SETIO   $$$´,0,0,0,0
        DB      0,´SETIO   COM´,0,0,0,0

; flags and address storage

CFLAG:  DB      0
IOBYTE: DB      0
ALTFLAG:DB      0
LDEVNUM:DB      0

; buffer(s)

IBUF:   DB      80H
        DS      80H

        END
```









Be your own chartmaker.

All it takes is GrafTalk™ and a little plain English.

We think GrafTalk is one of the most ingenious programs to come along since the birth of the microcomputer.

It turns out graphic designs, and text, so easily you can do it on your first try. And you don't have to be a computer expert.

Pie charts, bar charts, symbol and mixed charts, all accomplished on your desktop computer through a few simple instructions in plain English, all printed out on your printer.

No more handing over your figures to others. Then waiting around to see how the charts come out. Then waiting again. For revisions. And no more design expenses.

Further, in a bigger sense, GrafTalk helps you know your business better. For it lets you visually plot everything that's happening—share of market, quality control, sales trends, name it and you have it.

Not just when you have the time or budget. But anytime. GrafTalk puts it all at your fingertips. In color. Or in black and white. Depending on your equipment.

We sell and distribute GrafTalk worldwide, along with more than 200 other programs, all usable with most microcomputers on the market. We are, by a wide margin, the world's biggest single source of micro and minicomputer software—with offices in the U.K., Switzerland, West Germany, Sweden, France and Japan. And a nationwide Technical Hot Line that you can depend on.

Just give us a call at (212) 860-0300. Or fill out the coupon.

Lifeboat Associates
1651 Third Avenue
New York, New York 10028
TWX: 710-581-2524 (LBSOFT NYK)
TELEX: 640693 (LBSOFT NYK)

□ Send me information on GrafTalk.
□ Send me The Complete Lifeboat Catalog.

Name Title

Company

Street

City State Zip

Phone

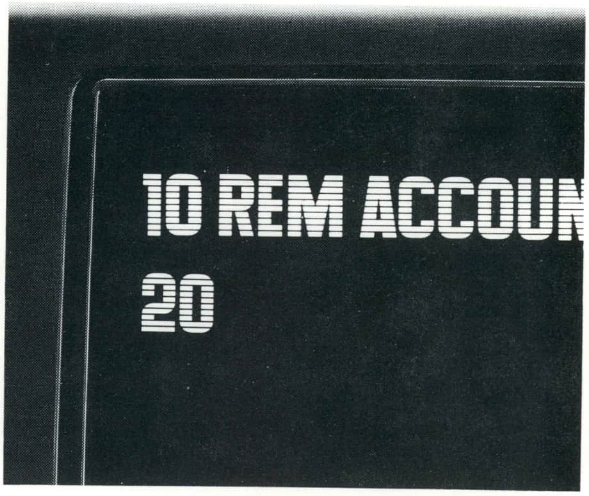
Lifeboat Associates

World's foremost software source.

Copyright © 1982 by Lifeboat Associates
GrafTalk,™ Redding Group, Inc.

REDDING GROUP INC.

BOY, IS THIS COSTING YOU.



```
10 REM ACCOUN
20
```

It's really quite basic: time is money.

And BASIC takes a lot more time and costs a lot more money than it should every time you write a new business software package.

Especially when you could speed things up with dBASE II.

dBASE II is a complete applications development package.

Users tell us they've cut the amount of code they write by up to 80% with dBASE II.

Because dBASE II is the high performance relational database management system for micros.

Database and file handling operations are done automatically, so you don't get involved with sets, lists, pointers, or even opening and closing of files.

Instead, you write your code in concepts.

And solve your customers' problems faster and for a lot less than with BASIC (or FORTRAN, COBOL or PL/I).

dBASE II uses English-like commands.

dBASE II uses a structured language to put you in full control of your data handling operations.

It has screen handling facilities for setting up input and output forms.

It has a built-in query facility, including multi-key and sub-field searches, so you can DISPLAY some or all of the data for any conditions you want to apply.

You can UPDATE, MODIFY and REPLACE entire databases or individual characters.

CREATE new databases in minutes, or JOIN databases that already exist.

APPEND new data almost instantly, whether the file has 10 records or tens of thousands.

SORT the data on as many keys as you want. Or INDEX it instead, then FIND whatever you're looking for in seconds, even using floppies.

Organize months worth of data in minutes with the built-in REPORT. Or control every row and column on your CRT and your printer, to format input and output exactly the way you want it.

You can do automatic calculations on fields, records and entire databases with a few keystrokes, with accuracy to 10 places.

Change your data or your entire database structure without re-entering all your data.

And after you're finished, you can protect all that elegant code with our run-time compiler.

Expand your clientbase with dBASE II.

With dBASE II, you'll write programs a lot faster and a lot more efficiently. You'll be able to write more programs for more clients. Even take on the smaller jobs that were out of the economic question before. Those nice little foot-in-the-database assignments that grow into bigger and better bottom lines.

Your competitors know of this offer.

The price of dBASE II is $700 but you can try it free for 30 days.

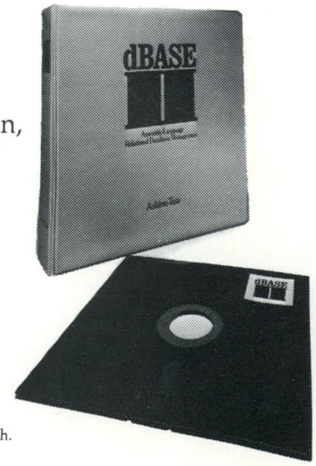
Call for our Dealer Plan and OEM run-time package prices, then take us up on our money-back guarantee. Send us your check and we'll send you a copy of dBASE II that you can exercise on your CP/M® system any way you want for 30 days.

Then send dBASE II back and we'll return all of your money, no questions asked.

During that 30 days, you can find out exactly how much dBASE II can save you, and how much more it lets you do.

But it's only fair to warn you: business programmers don't go back to BASIC's.

Ashton-Tate, 9929 Jefferson,
Los Angeles, CA 90230.
(213) 204-5570.



Ashton-Tate

©Ashton-Tate 1981

Also available from Lifeboat Associates.

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

LIFELINES™/The Software Magazine™
1651 Third Avenue, New York, New York 10028

Second Class Postage Paid
At New York, N.Y.

[redacted] 0