The Software Magazine™

$2.50 May 1982 Volume II, No. 12 (ISSN 0279-2575, USPS 597-830)

On TURBODOS

A Spelling Program

Bit Manipulation In PL/I-80, Part 2

Full Screen Program Editors: PMATE

Developing Applications With dBASE II

A Detailed Description of PLAN80, Part 1

8080 Assembler Programming Tutorial: Pitfalls of Programming

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.

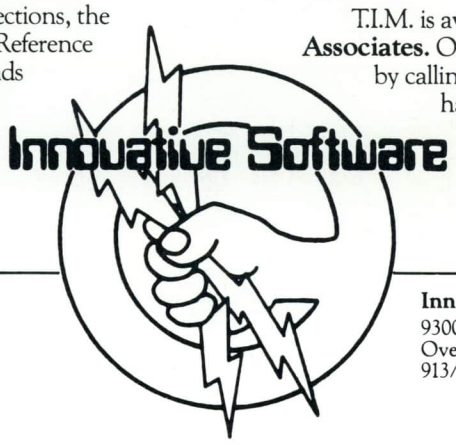
Features of T.I.M.

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 charactars 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.



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

AS AN INTRODUCTION TO OUR COMPANY

MAGTEK MEDIA IS PROUD TO OFFER .....

MEMOREX

FLOPPY DISKETTES

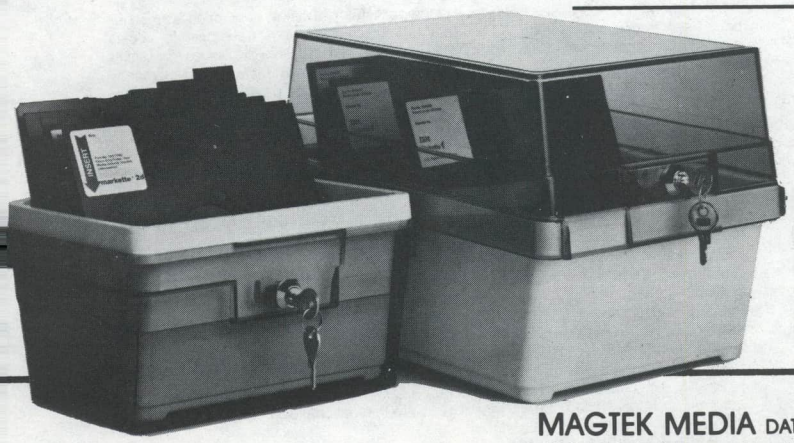
Product Family	Product Description	Part Number (3201-)	Price Per Disc $
8" Flexible Disc 1s Single-Headed Drives Single-Density Media	IBM Compatible (128 B/S, 26 sectors) Shugart Compatible, 32 Hard Sector Wang Compatible, 32 Hard Sector w/Hub Ring	3062 3015 3087	2.98 2.98 3.69
8" Flexible Disc 1d Single-Headed Drives Double-Density Media	IBM Compatible (128 B/S, 26 sectors)	3090	3.83
8" Flexible Disc 2d Double-Headed Drives Double-Density Media	Soft Sector (Unformatted) 32 Hard Sector, Shugart Compatible	3102 3181	4.47 4.47
Mini Flexible Disc 1d-40 5¼" Single-Headed Drives Double-Density Media 40 Track Tested	Soft Sector, w/Hub Ring 10 Hard Sector, w/Hub Ring 16 Hard Sector, w/Hub Ring	3481 3483 3485	3.12 3.12 3.12
Mini Flexible Disc 2d-40 5¼" Double-Headed Drives Double-Density Media Tested For 40 Track Tested Per Side	Soft Sector, w/Hub Ring	3491	4.54
Mini Flexible Disc 2d-80 5¼" Double-Sided Drives Double-Density Media 80 Track Tested Per Side	Soft Sector, w/Hub Ring	3501	5.54

I.B.M. MAXELL C.D.C. BASF & VERBATIM also available.

Please call us for all your computer needs.

Professional Protection for Floppy & Mini Disks

	Capacity	Features	Article Number	Length	Breadth	Height	
8" Standard Disks	90	Tray-with lid and lock 9 card rests	F90	13¾" app. (350 mm)	9.5" app. (240 mm)	9" app. (230 mm)	**$59.00**
	40	Tray-with lid and lock 4 card rests	F40	8¼" app. (210 mm)	9.5" app. (240 mm)	9" app. (230 mm)	**$42.50**
5" Mini-Diskettes	90	Tray-with lid and lock. 9 card rests	M85	13¾" app. (350 mm)	7⅛" app. (180 mm)	6.5" app. (165 mm)	**$46.00**
	40	Tray-with lid and lock 4 card rests	M35	8¼" app. (210 mm)	7⅛" app. (180 mm)	6.5" app. (165 mm)	**$29.50**



- Lockable — Portable — Secure.
- V shape for easy retrieval.
- All boxes with dividers in stepped pattern for indexing system.
- Self adhesive protective indexes supplied with boxes.
- All dividers are height of disks — eliminating need for guide cards.
- Dividers have locating device which permanently locks divider in position.
- Made from ABS Plastic — anti static.
- International standard size to fit most suspended frame systems.

MAGTEK MEDIA DATA SUPPLIES, LTD.
5916 - 18th Avenue, Brooklyn, N.Y. 11204
(212) 232-7166 1-800-221-0869 (Out of N.Y. State)

LIFELINES
The Software Magazine

May 1982

Volume II, No. 12

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

Lifelines (ISSN 0279-2575, USPS 597-830) is published monthly at a subscription price of $18 for twelve issues, when destined for the U.S., Canada, or Mexico, $40 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
The Software Magazine

May 1982 Volume II, No. 12

Contents

Opinion
Editorial Comments

Edward H. Currie

Bits, Bytes, Books and Random I/O

A number of you have written and inquired about the possibilities for standardization in the area of applications packages. For example, it has been suggested that wherever possible, control sequences should have a common definition among all applications. Of course, this is an excellent suggestion and you should continue to point out such areas where standardization reduces complexity for the end user. Some of us tend to forget that those just entering the arena of the microcomputer have a true appreciation of the need for uniformity and consistency. While in some respects this particular suggestion would require a monumental effort in an industry largely devoid of standards, it is important to maintain an awareness of ergonomics.

After all, many of us got our first inkling of the complexity of microcomputers when we discovered that there are eight ways to put a diskette into a disk drive – most of which are not very interesting.

Fortunately a number of fine texts are emerging to assist you in learning about micros and the multitude of applications now available. Arthur Naiman has provided a particularly nice discussion of WordStar, including chapters such as "Getting Started with WordStar in One Hour". This excellent book is well illustrated and reflects the author's concern with clarity.

Susan Blumenthal has authored a book titled "Understanding and Buying a Small-Business Computer", investigating such topics as software evaluation, consultants, peripherals, fundamentals of microcomputer systems, system selection, time-sharing, word processors, backup procedures, etc. This treatment manages to cover these topics while maintaining an acceptable level of complexity for the first time user.

X.T. Bui has provided an interesting examination of decision models, forecasting models, investment models and multicriteria decision-aid models for those interested in financial modeling and forecasting. His book, entitled "Executive Planning with BASIC", provides the listings for the models presented, along with descriptions of the methods employed in each model, illustrative examples and discussion of the results produced by the various models.

Those interested in Pascal will be pleased with the "Pascal Primer" by David Fox and Mitchell Waite. You will recall that these folks brought you that fine text the "CP/M Primer". This latest book was designed for those with a passing knowledge of BASIC and a wish to learn Pascal. The latter allows the programmer to produce programs which are from seven to ten times faster than their BASIC counterparts and 50% faster than an equivalent FORTRAN program. Their treatment uses University of California at San Diego (UCSD) Pascal as the version of Pascal discussed. Perhaps the best statement of the authors' intent is their prefatory remark that "The book is committed to the mastery of Pascal without tears".

It's interesting to note that application software is in fact the guiding force in determining the architecture of microcomputers, as manufacturers move heaven and earth to gain access to the wealth of software required by the market.

A number of manufacturers are configuring dual processor systems which permit the running of eight/sixteen-bit software. This seems to be an extension of the philosophy behind the Microsoft Softcard for the Apple. It may well be that future generations of machines will have multiple processors to permit the running of eight- and sixteen-bit applications. For example, there is a new card for the Apple which provides an 8088 with 64K of RAM. Look forward to seeing this trend spread across all classes of microcomputers in the near future.

Perhaps future generations of microcomputers will all use a common processor which can arbitrarily select microcoding, allowing the same processor to support a variety of machine codes.

Those who would attempt to demean sixteen-bit processors like the 8088/8086 have failed to recognize the fact that there are many benefits other than sixteen bit architecture and expanded instruction sets.

It should be noted that a major advantage of the sixteen bit micros is their ability to address large amounts of memory. This means that, for example, the manipulation of large arrays will now be quite practical. Those of you engaged in scientific endeavors (which often involve large data sets and programs written in FORTRAN) should be enthusiastic about this development, particularly in light of the 8087 arithmetic processor rumored to be available soon for the IBMPC. Godbout, Seattle and others are also configuring hardware which permits the use of this interesting device.

Also there seems to be some confusion as to the differences in the 8088 and 8086 at the software level. If you have come up with some neat techniques for determining, via software, whether the processor is an 8088 or 8086, send them along and we will publish them.

Some of you have suggested that perhaps the next quantum leap in computer technology will occur when biological logic circuits are available. Experiments are already underway to develop techniques for putting conductive pathways onto protein molecules. Once this technology has been mastered it may be possible to produce biological devices which are in fact tiny microcomputers requiring no external power sources but instead gathering necessary power from thermal energy available in the environment.

Furthermore these devices could be grown using techniques developed by genetic engineering. Thus future computers could be grown to specification as true "micro" computers.

The mind boggles as we contemplate what the future of microcomputing holds for us all...

ON UNIX

Jean L. Yates

The Future of UNIX, Part 2

Who Uses UNIX?

Gnostic Concepts is still in a preliminary stage of data collection for the market numbers of its UNIX survey, but we have some approximate data. We are estimating the number of source and binary sites for UNIX, as shown in Figure 1. The Bell System dominated until 1981, but almost 4,000 commercial binary licensees in 1981 resulted in commercial sites overtaking the Bell System for the first time. What may be a little surprising is the large number of universities and research institutions using UNIX. We suspect that almost ninety percent of universities with a computer science department have a UNIX source code license. The government/military installations grew about twenty-five percent this year and will still remain a small but lucrative proportion of the total UNIX user community.

Who are the UNIX users today? Let's take a look. In education there are about 715 source code licenses shared among 1,720 sites. Government has approximately 125 source code licenses among 270 sites. Commercial has approximately 350 source code licenses among 4,000 sites. These numbers must be tempered by the fact that the new reporting period for binary licenses will not occur until several months into 1982. At that point, if the Bell System chooses to release the number of binary licenses sold in 1981, we may find that these numbers are underestimates.

Commercial users now include independent telecommunications companies, computer and software companies, large industrial corporations, OEMs selling to vertical markets and end users. The commercial end users will increase more rapidly, while computer and software companies' rate of increase begins to lag as they are saturated. These companies were some of the first to purchase UNIX in the commercial environment, because they are using it to develop products that they will send to end users.

The Bell System

In the Bell System, about 5,000 computers employ UNIX. They are not only used in research and software development, but utilize variants of UNIX for operating equipment and are also used as office tools. The Bell System stays one version of UNIX ahead of the outside world; they are now running System IV internally. Some installations are rumored to be running Berkeley's VAX version BSD 4.1. This is a version not supported by Bell Labs, but supported by Berkeley.

It is important that the Bell System has internally standardized on System III as a commercial as well as a research product. Although UNIX was previously a viable product from a research sense, moving UNIX into a commercial realm indicates an acceptance (internal to the Bell System) of the product as an operating organizational standard, rather than just a research tool. This indicates that the product should have a long and fairly consistent lifeline.

Government/Military

The government/military installations of the Department of Transportation, the Army's DARCOM facility, the Navy at Bethseda and the Air Force at Gunther are small, but indicate a trend towards the use of UNIX in the military. The Bell System has now announced UNIX support at these installations in the military sphere.

Berkeley UNIX

We've mentioned Berkeley UNIX many times here, and I'm sure that experienced readers know all about it, but I'm going to review it for those of us that are not so familiar with it. In 1975-76 Ken Thompson went to Berkeley and worked there as a visiting professor for a year. His efforts, along with the efforts of Dr. Fabry, Bill Joy, and many other programmers, resulted in Berkeley's version of UNIX. Today this version is preferred by many commercial and research installations over the Bell System's V7. Berkeley has two flavors of UNIX, 4.1 and 2.8. Berkeley 4.1 runs only on the VAX computer, although with judicious use of porting techniques, it can be ported to the 68000 microprocessor.

Berkeley has shipped 230 tapes containing BSD 4.1, but we estimate that there are many more installations than represented by those tapes. Portions of Berkeley's code, specifically the C Shell, vi, and some of the UNIX utilities, are used widely across the UNIX community. Vi is probably the most popular text editor for the UNIX operating system today. At Berkeley, most of the refinement and optimization of UNIX code occurred on the VAX computer and is represented in BSD 4.1. This is probably some of the cleanest and most clearly optimized UNIX code available today, and people within and outside the Bell System utilize it.

Berkeley also continues to sell BSD 2.8, UNIX for the PDP-11. They have shipped over 100 tapes of that product. It contains many of the same elements of 4.1, but in a less optimized form.

Support

The key to using UNIX in a commercial environment is the ability to support the product. In the past, universities could justify internal support costs by the low initial purchase price. Within the Bell System, individual installations supported themselves with the aid of Bell Labs' research and hotline facilities. The government/military, commercial installations, etc., all supported themselves, or in many cases

(continued next page)

turned to Berkeley for advice. Today, the support situation has changed. Bell Labs still supports Bell Labs while ATT/Western Electric supports itself and the government/military. Microsoft presently supports the PDP-11 with its XENIX product and Altos Computers. Microsoft has many products under development. Interactive Systems supports the PDP-11 and the Onyx; I believe they have started support for Plexus.

Unisoft of Berkeley, which is Jeff Freedman's company, supports Codata, CM Technologies, and Dual Systems. Unisoft only supports the 68000 version of UNIX and also performs ports of UNIX to the 68000 microprocessor. Berkeley supports Berkeley itself, and supports many tie-ins across the Arpanet. It has an informal support network of all purchasers of 2.8 and 4.1 tapes. Formerly, this support consisted of sending electronic mail or calling the professors and programmers at Berkeley. How long they will have time to support the growing number of Berkeley UNIX users is unclear, but obviously the situation can't go on for much longer. Wollongong supports Perkin-Elmer.

Then we see a number of computer companies that either are developing UNIX-based products or use UNIX internally as a standard, and mostly they support themselves. One group of concern is large installations of UNIX in industrial corporations, for example, aerospace companies or manufacturing organizations. For the most part, they support themselves, with the aid of Berkeley.

Supporting UNIX

What does it take to support UNIX? Today, Bell Labs uses a telephone hotline for its clients, as do many commercial vendors. Interactive Systems uses an innovative and apparently effective mail system where users send messages to the support team and they respond in like. Berkeley has supported quite a few people via their electronic mail on the Arpanet, and this will continue in an informal sense. At some point obviously, the support issues will grow beyond what Berkeley can handle. It's not that there are so many bugs in 4.1, because they aren't; but lots of beginners ask lots of questions. A big wild card in the support situation is DEC. The majority of computers out there right now running UNIX are DEC equipment, but the scenario will change rapidly. However, it's unclear that DEC will support UNIX. My message to DEC is, "Please, please DEC, it will make life so much easier for us and you could make so much money."

There has been some speculation that the Bell System might eventually support UNIX for the outside world, but obviously this is a long range scenario. The announcement of support for the government/military is an indication of a trend in that direction but the cost and personnel implications of supporting a possible 10,000+ installations of UNIX by late 1982 are mammoth. At best, we speculate that the Bell System will suppport source code licensees only, leaving binary licensees' support to the source code vendors with resale privileges.

More and more hardware companies are turning to software houses with expertise in UNIX for their support needs. Interactive Systems supports Onyx and now is beginning to support Plexus. Unisoft of Berkeley is supporting about four different companies with 68000 based systems. Microsoft, with the XENIX product, is supporting PDP-11 installations that produce and will soon be supporting Altos and many other companies. The level and type of support, etc. varies from facilitiy to facility but all have themes—a combination of telephone inquiry, electronic mail inquiry, and a team of support engineers answering questions. The support issue is tying up hardware companies, as is the issue of porting.

Who's Looking at UNIX?

Mainframe companies are looking at UNIX; we speculate that companies ranging from IBM to Amdahl, NCR, Univac, Honeywell, and Prime are looking at the operating system; some have released products. In the minicomputer area, we hope that DEC is looking at support for UNIX along with Data General, Perkin-Elmer, and Hewlett-Packard(again speculative). In the small business and personal areas, Basic Four, Xerox, BBN, Convergent Technologies, many OEMs, Apple, Fortune Systems, and companies such as Instrumentation Laboratories all either are looking at or have released products based on UNIX.

In Japan, software now is understood to be the key to entering the microcomputer market, and just about everyone is looking at UNIX as a potential way to enter the U.S. market with a large software base. Significantly, microprocessor manufacturers are looking quite logically at UNIX with Zilog, in our opinion the furthest along towards fully supporting and selling a UNIX based product—ZEUS. Zilog has had a released product for about six months now, and their documentation and support has progressed by far the furthest of the three companies. Motorola has yet to announce any type of UNIX support but it is strongly believed that they are working on it; the same situation occurs with Intel, although their recent blitzes of advertisements for their Intel-developed operating systems may indicate a possible desire to stop the UNIX onslaught.

Porting UNIX

The porting of the UNIX operating system onto a new microprocessor-driven system is not a simple proposition, as many companies can testify. The cost can range from a minimum of $25,000 to over $1 million, depending upon the decision as to the completeness of the port, the company doing the port, and the target processor. Porting UNIX can take three months for a team of skilled experts working quickly, to two years for a group of less experienced people trying hard but learning as they go along. Also, there are various levels of implementation of UNIX, specifically implementation of VAX versus PDP-11 Berkeley UNIX. If VAX 4.1 Berkeley UNIX is needed, fewer shortcuts are taken in the implementation of 68000 based systems. I will note here specifically that Unisoft of Berkeley offers a port allowing the use of the BSD 4.1 versions of products such as the C shell and vi. This is a distinct advantage for a vendor wanting to offer maximum capabilities and quality of UNIX software.

Another consideration in porting UNIX is software compatibility. When the port is made, a lot of choices are made about how the system drivers operate and what kind of UNIX is port-

ed onto the target machine. Applications software portablity is still up in the air, but increasingly we find commercial vendors taking pieces of Berkeley UNIX while maintaining standardization with System III.

Getting a compiler that works correctly for a target processor is a difficult proposition, and only the best software houses performing ports have managed to get complete debugged cross compilers running reliably. Another factor which limits the speed of performing ports is the necessity of porting to prototype hardware. If the port is to a piece of untried hardware, which it seems is mostly the case these days, software vendors can spend as much time tinkering with the hardware as they do with the software.

Last and most important, we cannot emphasize enough the requirement for very skilled personnel in the porting job. Unfortunately, this isn't a job that can be done by throwing a lot of personnel and time at it. Highly trained UNIX people with hardware architecture expertise are needed to perform the port correctly and quickly. Unfortunately, Berkeley probably hasn't produced more than twenty UNIX "gurus"; of course there are other research institutions, but there still are not that many people with this level of training. For that reason, everyone has been slowed down in moving their software over, as they wait in line for the teams of qualified people to get to their machine.

As you can see from the list in Figure 2, there have been a number of UNIX ports completed. This list is not complete because there are many in progress and I'm sure I've missed them. But basically you can see in the mainframe, mini and micro world that the operating system is moving across a lot of places. By the end of 1982 we expect to have over forty more companies added to this list.

UNIX Applications

We now come to a section where I will attempt to describe some of the business applications that are either existing or planned for UNIX. Software development and engineering applications are in use, communications as well. These are some of the more traditional uses of UNIX, but at the top of the list are business applications, such as accounting, financial modeling, transaction processing, and data entry. The largest number of new products will come in the business area, as many different vendors start selling UNIX systems into business and office automation environments.

In Figure 3 I am showing you the way that Gnostic Concepts is counting UNIX users. We are attempting to quantify not only the people who are out there, but how much they are spending and how much they will spend. As you can see, we have personnel, outside services, packaged software supplies, and "other." This is not necessarily in scale for the proportions but it is approximate. Also, you see hardware and as you look at Figure 4, which is UNIX-related EDP expenditures, you see an expansion of the $9 billion figure that was quoted; this was really rather a misquote, since the quote came from the top end of the top number at the end of the chart, and did not describe the range.

The 1986 figure still gets up to $9 billion but that is not on this chart. What I've done here (and this is data that we are still in the process of developing) is try to give you some idea of the ranges of UNIX-related sales. As it stands right now, this chart includes UNIX and the "UNIX-like" products such as Cromix and Whitesmiths', Idris, etc. The white part of the bar is personnel, support, and outside services and, at this point, applications software. We have not yet separated applications software out of this pool; we have not reached that level of research, although we will shortly. The middle part of the bar is hardware, and as you can see, it will increase from about $7 million in 1980 to $1.8 billion in 1985. The bottom part of the bar indicates cost for UNIX, the system software, utilities and languages themselves rising fairly rapidly to a $250 million market in 1985.

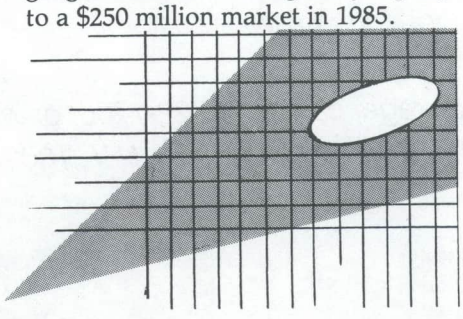


FIGURE 1 — WESTERN ELECTRIC UNIX LICENSING DATA

SEGMENTS	Total Number of Computers Installed* As of 8/25/80	As of 12/31/80	As of 5/1/81
BELL	630	750	825
UNIV	1336	1575	1716
FED GOV'T	156	197	216
COMMERCIAL	224	287	341
TOTAL	2366	2809	3098

*Source Lic Data from USENIX Jan and June 1981

1980-81 65 percent increase

SOURCE LIC'S*

	Source Lic's*	Computers Installed
12/31/80	800	1600
5/1/81	1078	2273
9/1/81	1249	2538

*Source Lic Data from Otis Wilson Jan 12, 1982

1980-1981 106 percent increase

FIGURE 2 — SUCCESSFUL UNIX PORTS

- AMDAHL 470 (IBM 370)
- BBN C50/C70
- DATA GENERAL (C COMPILER)
- DEC PDP11
- DEC PDP7
- DEC PDP8
- DEC VAX
- FORTUNE SYSTEMS
- HONEYWELL 6000
- IBM 370 (C COMPILER)
- IBM 4300 (under development)
- IBM SERIES 1
- IBM 8885/8086 (under development MICROSOFT/ALT/OS)
- MOTOROLA 68000 (under development — 40 companies)
- ONYX 8002
- PERKIN-ELMER
- PLEXUS SYSTEM 10
- ZILOG Z8000'S (under development — 15 companies)

(continued next page)

FIGURE 3 — USER SPENDING CATEGORIES

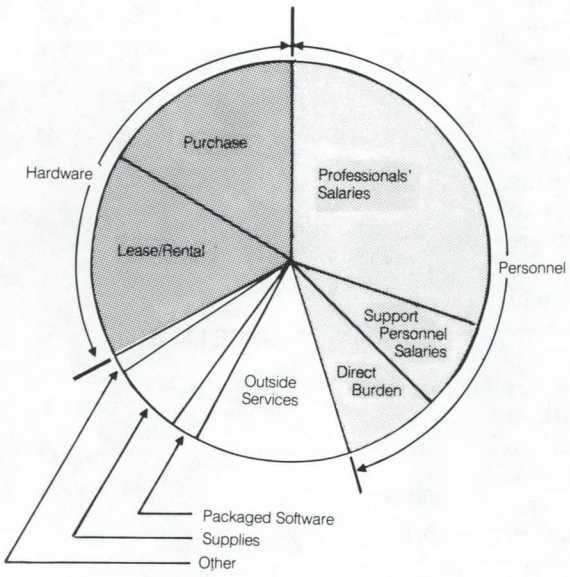
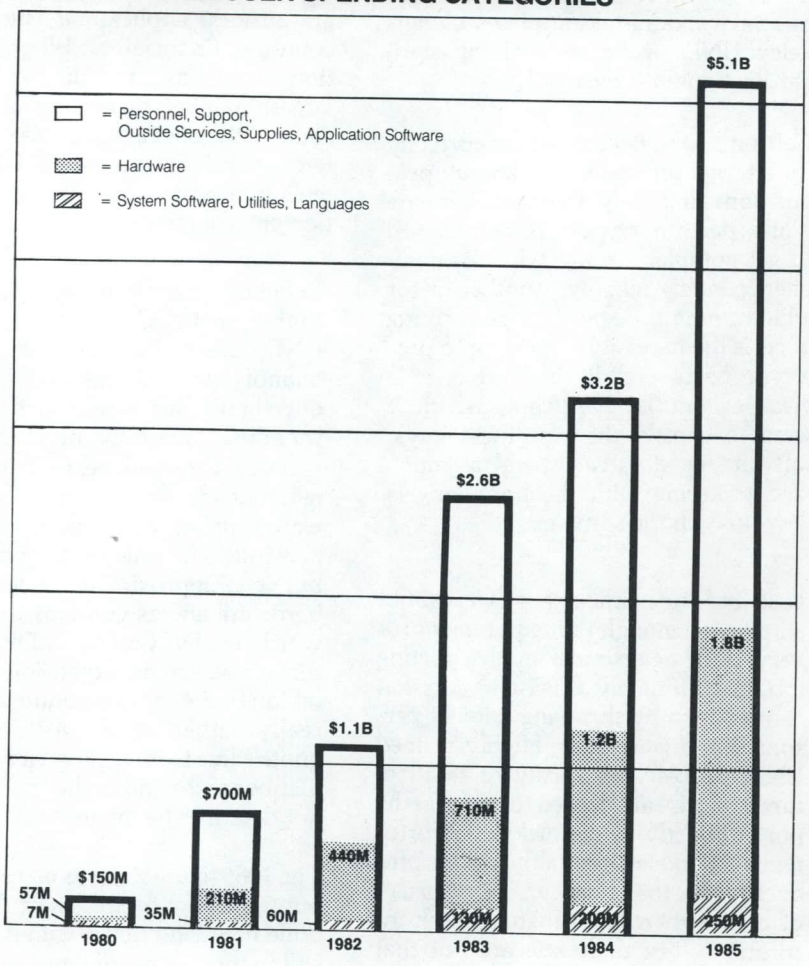


FIGURE 4 — GNOSTIC CONCEPTS SURVEY OF UNIX USER SPENDING CATEGORIES



A Spelling Program

Harry Tennant, Ph.D.

Touch panels, light pens and joysticks notwithstanding, most human-computer interaction is through a keyboard. And frequently what goes through the keyboard is misspelled. Fortunately, most of the spelling errors and typos that are generated when interacting at a keyboard are noticed and corrected immediately. In fact, if you are like me, the most frequently pressed key on the keyboard is the backspace key! Some spelling errors, however, get through. Whether it is carelessness, ignorance or indifference, the effect is that the computer swallows a lot of misspelled words.

In some application areas it doesn't make much difference to the computer how badly one might spell. In text editing, for example, the computer doesn't care about spelling. The user will be embarrassed by the errors, not the computer. In other applications, however, ones in which a program interprets what is typed, if the user doesn't spell the input correctly, the program won't have an inkling of what the user was trying to get across.

Spelling is a common task, and misspelling (believe it or not) is something that people do in a more or less uniform way. In other words, spelling correction is a task that is well suited to computerization. This article will discuss some of the attempts at spelling correction programs and will indicate some of the trade-offs in designing spelling correction algorithms.

Several spelling correction techniques will be discussed. One algorithm will be described in detail including an assembly language implementation and some performance indicators.

Before getting into detail, allow me to illustrate the importance of spelling correction. I was once on a research team writing a natural language understanding system for the military. Visitors from the military would come by for demonstrations from time to time. Many man-years of effort had gone into the development of this system in an effort to improve its language understanding capabilities. In addition, about two or three days worth of effort had gone into a spelling correction routine to correct errors in the users' typed queries. During one demonstration, the demonstrator inadvertently misspelled a word, and didn't notice his mistake. When the program noticed and corrected the spelling error, the reviewer was so impressed, he kept mentioning it throughout his visit. No mention was made of the natural language understanding capabilities, but that one spelling correction left him very favorably impressed with the project!

Spelling Correction in Document Preparation

One useful application of spelling correction is as an aid in document preparation. The spelling checker must include a large spelling dictionary and a fast checking algorithm. A user submits a file of text to the spelling checker, then the checker looks up each word in the dictionary. If the word is found, it is accepted. If it is not found, it is analyzed as a possible misspelling. Depending upon the confidence one has in the program, the correction may be automatically substituted into the file, or the user may be consulted on the matter.

Ralph Gorin at Stanford University wrote a program in the early '70s to do spelling correction on text files. (It has since been improved upon elsewhere.) His goals were to write a fast spelling checking algorithm that would operate over a large vocabulary. The basic datastructure is a three dimensional array of 26x26x10 elements. Each element in the array is the head of a list of dictionary words. An array element represents all the words in the dictionary that start with a particular letter pair and are of a given length. When the spelling program reads in a word, it looks it up in the array. If it finds it there, it goes on to the next word. If an exact match is not found, the program attempts to detect and correct one of four specific errors. They are:

1)1 wrong letter	(mixspell)
2)1 missing letter	(mispell)
3)1 extra letter	(misxspell)
4)2 transposed letters	(mispsell)

This spelling corrector was implemented in DEC10 assembly language so it would be fast, and parts of the algorithm are very dependent on particular DEC10 instructions. Some of the efficiency of the algorithm derives from the fact that many of the characters of two words can be compared simultaneously with single operations on the DEC10's 36 bit words. The check for one wrong letter is straightforward. The check for one missing letter is done by inserting a null (0 character) in each character position in the word, then checking for one wrong character. The check for an extra character is made by deleting each character in turn and looking the new word up. Transposes are checked by transposing each pair of adjacent characters in the word and looking up the resulting new word.

The most obvious difficulty with this kind of system is that it must have a large vocabulary. Programs with vocabularies in the hundreds of thousands of words are not uncommon. However, one technique to reduce the vocabulary size requirements without diminishing the number of words that are recognized is to do morphological analysis - automatic recognition and stripping of prefixes and suffixes. This is a trade-off of processing time (to do the morphological analysis for an unrecognized word) vs. storage (equivalent to a vocabulary as much as 5 to 8 times larger). It is also a convenience for the dictionary builder, and insurance against omitting a morphological variant by oversight. Of course, some additional checking is also needed to avoid accepting variants of irregular words like "feets" and "goed".

(continued next page)

A warning is given to users of this spelling program, at least in the version I use, against assuming that a document will be free of spelling errors if the words in the file have been checked against a dictionary. Fairly frequently, misspellings result in a character string which is also a properly spelled word, but does not belong in its context. A quick search of some uncorrected files yielded many examples, a few of which are shown here.

1) *Do* to the bad weather and lack of interest, the CSL canoe trip planned for this weekend has been cancelled.
2) *Its* a girl!
3) PS: <tennant>meeting.report contains some observations *form* the June meeting
4) How many hours of NOR were *thee* for each type plane during the years 1970, 1971, 1972, and 1973?
5) *Think* you, I think I have found the answer.
6) If it chooses *wrong*, it must backup and try over.
7) One of the parses will eventually be abandoned *latter* in the parse when more information is available.

Detecting errors of this kind requires more knowledge of the context in which the "respellings" occur. Detecting many such errors is within the state of the technology of natural language understanding.

High Speed Word Recognition

A program was developed for the PLATO educational computer system to quickly search a dictionary to verify that either a student's input word was in the dictionary or it was a misspelling of a word in the dictionary. In this application, the goal was only recognition of words, not replacing misspelled words with their proper spellings as in the application discussed above. In the PLATO scheme, the correct spellings of the words are not even stored in the system, so correct spellings cannot be substituted for misspellings!

The PLATO algorithm works by representing not the spellings of dictionary words, but features of the spellings of dictionary words. The most important features for word recognition occupy the most significant bits of the computer word. The less significant features occupy the less significant bits. The feature words were sorted, so the words with the most similar sets of word recognition features were closest together. A user's word can be checked against the dictionary words by 1) building a feature word for the user's word with the same algorithm used for the words in the dictionary, then 2) doing a binary chop search of the dictionary feature words for a match. If there is an exact match, the user's word has been found in the dictionary. If there is not an exact match, the words closest to the point where the search terminated are the most likely candidates to be the proper spelling of the user's word. The degree of disagreement of the features of the user's word and the nearest dictionary words is computed. If the disagreement is sufficiently small, the user's word is accepted as a misspelling. If the disagreement is over a fixed threshold, the word is rejected as unknown. Disagreement was computed by EXCLUSIVE ORing the unknown feature word with a dictionary feature word, then counting the number of ones (the number of disagreeing bits) in the result.

The features used in the PLATO word recognition routine are shown below in order of importance.

1) length
2) first character
3) set of letters
4) set of digraphs
5) syllables

The algorithm was implemented on 60 bit per word CDC computers, so there was plenty of room for features in each computer word. Each dictionary word corresponded to one computer word for efficient comparison. The first letter was stored in ASCII form. The length was represented in grey code so that words of nearly equal length would be in the same general locations in memory. The letter content was represented in a 16 bit field. If there was at least one "a" present somewhere in the word, the bit corresponding to "a" was set. Representing a set of 26 letters in 16 bits required some overlap – some bits indicated more that one possibility, e.g. one bit indicated that there was either an "s" or "q" in the word. The set of digraphs (letter pairs) in the word was a redundant mapping like the letter content mapping. Digraph content was included because of the frequent error of letter transposition. Finally, an attempt was made to represent the syllabic content of the words so that words that sound the same would have similar representations. Ideally, one would like to be able to notice a similarity between "chevrolet" and "shevrolay". An approximation to syllable division was made by mapping consonant-vowel pairs. The syllables in "california" by this method were ca li fo ni. Similar sounding consonant-vowel pairs were mapped together. As the reader might expect, this feature was not wildly successful, but it is certainly an interesting idea which deserves further consideration.

SPELL in The Stiff Upper Lisp

The last spelling correction algorithm I will present is the one that is included as a built-in function in The Stiff Upper Lisp. The algorithm is a variation on the algorithm for spelling correction in Interlisp. The algorithm compares an unknown word to successive members of a candidate list of known words. It differs from the other two algorithms presented here in that it is assumed that the unknown word is certain not to be an exact match to any dictionary word. The check for an exact match is made immediately when any word is read into Lisp.

As the unknown word is compared to a candidate, on a character by character basis, demerits are accumulated for disagreements. When the ratio of demerits to the length of the unknown word gets higher than a threshold, the unknown word is proclaimed not to be a match for the current candidate. Comparison fails at that point. If the unknown word and candidate word compare each character without accumulating too many demerits relative to word length, they are proclaimed to match.

The kinds of disagreements between words that are likely to be generated from spelling errors do not cause demerits. The acceptable disagreements, as with the above algorithms, are as

follows. In the SPELL function, more than one error can be corrected in the same word.

1) transposed letters
2) single letter errors
3) single omissions
4) stuttering (repeated characters)

When the current character of the unknown word does not match the current character of the candidate word, the immediate environment of the disagreement is examined to see if one of these four acceptable errors is the cause of the disagreement. If so, no demerits are given. If not, a demerit is given.

In the actual implementation, computing the demerits to length ratio and comparing that result to a threshold of disagreement would be unduly inefficient. Instead, when the spelling correction routine is entered, an integer (called the grace) is computed which indicates how many demerits can be accumulated before a match to a candidate word is rejected. All unknown words start off with a grace that depends upon their length. Words of nine characters or more have an initial grace of four. Five to eight characters are given a grace of three, less than five get a grace of two. Before a character-by-character comparison of the unknown word with a candidate, the difference in length between the unknown and the candidate is computed and subtracted from the grace. If the resulting value of grace is less than one the candidate is rejected. Otherwise, character by character comparisons begin, and after each one the grace is checked. If it falls below one the candidate word is rejected.

This kind of spelling correction is designed to be used by Lisp programmers. It would be most effective in a dynamic environment in which the list of candidate words may change radically from moment to moment, but at any particular point in time, the list is relatively small. An example of its use is in natural language understanding programs. In the sentence,

> The slithy toves did gyre and gimble in the wabe,

the list of candidates for the unknown words can be restricted by their apparent lexical classes (noun, verb, adjective, etc.). "Slithy" must be an adjective or noun (the "y" suffix suggests an adjective). The same is true for "toves" (the "s" suffix suggests a noun), but since it preceeds the verb "did", "toves" must be a noun. "Gyre" and "Gimble", coming after "did" are probably both verbs. "Wabe", since it follows "the" is a noun or adjective, but because it ends the sentence, it must be a noun. With these restrictions, an unknown word need only be compared to the lists of nouns, verbs, or whatever is appropriate. Further restrictions can be added if meanings are taken into account. In the sentence,

> I had a fried glerk for breakfast,

not only can we guess that "glerk" is a noun, but that it is a kind of food. Of course, there comes a crossover point where the effort required to further restrict the list of candidate words is greater than the effort required to check them. In many situations, however, this kind of information is already available, and it must only be used.

How Well Does SPELL Work?

It is difficult to describe how well a spelling corrector works. What we would like to see it do is correct all and only those misspellings that a human would correct. To give some idea of performance, short of testing a collection of people and comparing that to SPELL, I have done some quick tests. I include both the results and the test data, so the reader may judge the validity of the tests for himself.

To determine how well SPELL corrects misspellings, I located a list of the hundred words most commonly misspelled in one study of college freshmen. I generated misspellings for these words (only the correct spellings were provided), and ran them through SPELL. Out of 104 words, 87 were accepted (84%). Of course, a higher percentage could be had if the algorithm were made more forgiving of errors. The difficulty with this is that more inappropriate words would be accepted as misspellings. I invite the reader to peruse the list of words. (See Table 1.)

The subject of the second test was to determine if the algorithm is too loose. I took a sampling of the misspellings from the last test, then ran them through SPELL for the four words closest to the correct spelling in the dictionary. For example, "accomodate" is a misspelling of "accommodate". I looked "accommodate" up in the dictionary and found the two words before it ("acclivity" and "accolade") and the two words after it ("accommodating" and "accommodation"). The misspelling "accomodate" was then compared with these four close words. The results were that only three of the misspellings matched close words, out of 88 comparisons (3%).

I think these results seem a little too good. A differently designed test may have come out differently, and the sample is not very large. Besides, the tests are simply to give a feel for how well the algorithm works, and perhaps that, at least, has been accomplished.

Conclusion

Spelling Correction is a relatively simple process, as well as being fairly impressive. For a small amount of coding effort, a spelling corrector can add enormously to the convenience and apparent intelligence of an interactive program.

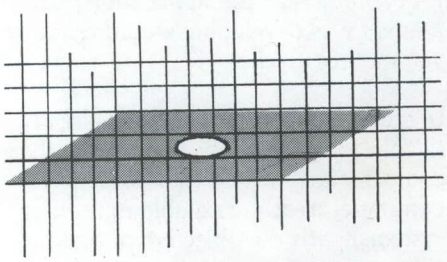
References

The Stiff Upper Lisp is available under CP/M from Lifeboat Associates (1651 Third Avenue, New York 10028) and under TRSDOS from Tennant and Tennant Computing (3537 Ridgemoor Drive, Garland, Texas 75042.)

Gorin, Ralph. Spelling Check/Correction Program. Program documentation. Available from DECUS. March 4, 1971.

Tenczar, P.J., W.M. Golden. Spelling, Word and Concept Recognition. Computer-based Education Research Laboratory, University of Illinois, Urbana, Illinois.

(continued on page 48)



On TURBODOS

Ron Fowler

This article is intended to acquaint the reader with a relatively new disk operating system for the Z80: TURBODOS. It is not a review in the strictest sense (although I'll pass on my opinions where appropriate), but will instead stress the functional characteristics of the system that will, hopefully, be helpful in making a decision when selecting an operating system.

I'll begin with a chronicle of events that led to my own involvement with TURBODOS, and continue with a short overview of the system and its capabilities. From there, I'll discuss the system in more detail, touching on each major feature incorporated in TURBODOS, along with a look at each of the major system utilities provided. A short discussion about potential problems that may be encountered will follow, along with a look at long-range plans for the operating system revealed in a conversation with the author of the system. I'll conclude with a short description for programmers.

Background

When I began work for my present employer (Willens-Michigan, a Detroit-based typesetting firm), the only software development system available for my use was a Z80-based S-100 machine, which had been purchased originally to automate the company invoicing process. At the time I began using the machine, it was being used more and more for secretarial word processing needs, and available machine time was diminishing. Within a few months, it became obvious to my employer that, in order to make effective use of my time, another computer would have to be purchased just for software development. Further, there was the probability that the applications programs I was developing would require yet another machine.

In order to make best use of the already sizable investment in equipment, the company assigned me the task of investigating currently available multi-user systems, with the objective of making as much use as possible of the existing Z80 system. The proposed system would have to accommodate a minimum of three users, with possible expansion to as many as five or more. A high priority was given to assuring a relatively low per-user expansion cost, after the initial investment. Yet another consideration was the need to run our existing CP/M software (WordStar, our invoicing program, and a number of others) on the new system. All in all, this was a rather formidable set of requirements to build around an 8-bit processor!

We initially investigated several multi-user systems, among them MP/M, OASIS, and ALTOS. For software compatibility reasons, the field was narrowed down to MP/M. We ordered system documentation, and made arrangements to visit some local MP/M sites. Our findings were rather disappointing; we found that MP/M response time suffered greatly with only a few users. Printer spooling[1] ran at a low priority, which caused background printing to stop entirely for long periods of time. There were (at that time) no provisions for file-lockout[2]. Finally, it seemed we would also have to spend a considerable amount of development time extending our CP/M *BIOS* for use under MP/M.

It was around this time that an article in *Kilobaud Microcomputing*[3] caught my eye: it was a comparison of multitasking versus multi-processor systems. An important point of the article was that complete single-board computers were being manufactured that plugged into the S100 bus. These boards were not bus masters, in that they appeared on the bus only as I/O ports (although an expansion board would allow one of these slave computers to become the master processor, so that a complete system could be built around this hardware). The hardware could be purchased either as a complete system, or as single circuit boards, the latter being more suited to our needs.

We then began making inquiries of various manufacturers of this type of hardware. The system mentioned in the *Microcomputing* article suffered from what I considered a glaring deficiency: the operating system provided was standard CP/M itself, with a special program running in the master processor to provide support to the slaves. I strongly felt that this new hardware concept should have an operating system that was designed with networking in mind, although I doubted that such an operating system would be at all CP/M compatible[4].

During our research, we found a company in Tustin, California (MuSys Corp.) selling a similar system, running a real network-oriented operating system which it called *MuDos*. It turned out that MuDos was being offered by a number of manufacturers, including Industrial Micro-Systems, under the name *TURBODOS*. The operating system was produced by a company called Software 2000, based in Los Alamitos, California.

The slave CPU boards sold by MuSys contained a full 64K of memory, a serial port for the console terminal, and a network port for communications with the S100 master CPU.

We again ordered documentation, and called several MuSys customers, all of whom responded favorably when asked their opinions of the system. Since MuSys Corp. would provide the disk drivers for the Morrow M26 (which we already had on order), we decided we had found our system, and placed an order for the full networking version of the operating system. We also ordered two slave microcomputers, one of which was the newly-developed Net-82, providing an extra 64K of memory (two banks), a vectored interrupt controller, and an optional floating point math processor.

I should point out that, since we had to interface this new equipment to our existing hardware, there was some software development work necessary to bring the system up. For example, the system had to read and write our DJ2D floppies, as well as communicate with

an IBM card punch, a PMMI modem, and our NEC daisy-wheel printer. Fortunately, driver specifications are provided with the system documentation, and I had most of the necessary drivers written by the time all of our components arrived. After a couple of evenings of debugging, we were able to put TURBODOS "on the air", a master operating system that communicated with our peripherals, the network, and our M26, slave operating systems for each of the slave CPU's, and an OS loader program (provided with TURBODOS, and configured by the user) to bring the system up.

We have had the system running for about six months now, and we're pleased with its capability. We've had no reliability problems, aside from a slight problem with the Net-82 CPU (which was quickly replaced by MuSys; they shipped a new Net-82 as soon as we notified them of the problem and *before* we sent the old board back). Further, the software is well-supported, both by MuSys and Software 2000[5].

The remainder of this article will be devoted to a detailed overview of the TURBODOS operating system.

TURBODOS Features

TURBODOS supports up to 16 disk drives, ranging from single-density floppies to hard disks in excess of 1000 megabytes per drive. Files may range up to 134 megabytes in length.

Additionally, the system supports up to 16 printers per CPU, and utility programs are provided for manipulating the various printers and printer modes.

Networking versions of TURBODOS may be configured, as either network masters or network slaves. As many as 16 slaves may be serviced by one master.

Full printer spooling and despooling is supported for up to 16 printers, both through utility programs, and extended BDOS calls. Sixteen communications channels are also supported for modems, inter-system channels, links to peripheral devices, etc.

No system tracks are required on any TURBODOS disk, since the operating system is loaded once (from a disk file on the boot-up disk) and remains resident until the next cold start-up. Also, since disk buffer sizes may be specified by the user, it is possible to read the disk without interleaving the sectors, resulting in a vast speed improvement.

The amount of disk buffer space may be changed dynamically using a system utility, so free memory can be used efficiently while speeding up disk operations. Also supported is a program load optimizer, which scans the disk allocation map to determine the contiguous sections of a program on the disk, then loads these sections in the most efficient order.

TURBODOS may be configured as multi-tasking (in fact, this is necessary in the network master), so that many functions can be performed in the background. Although the documentation doesn't help one do this, the information is available from Software 2000, and should be in the next documentation update.

The system supports logon and logoff utilities, with usage logging to a disk file. Passwords may also be specified as a login parameter.

Among the file attributes supported are the *SHARED* and *EXCLUSIVE* attributes, to control simultaneous access by more than one user. A file may also have a *FIFO* attribute (I'll explain that one more fully a little later), a *GLOBAL* attribute, and an *ARCHIVED* attribute (this is supported by the COPY utility, to allow flexibility in backing up large disk systems). The only CP/M compatible attribute is the read-only attribute. All attributes may be set or reset from high-level languages supporting a rename-file function, using special characters in the rename fields.

TURBODOS supports file interlocks at both the file and record levels. A unique record-locking capability allows locking declaration from any high-level language that can manipulate disk files.

The system includes a batch processor similar to the CP/M SUBMIT utility, but with enhanced capability. In addition, the console command processor allows multiple commands to be entered on a command line, separated by backslant characters. System calls support the command line processor, and allow command lines to be passed from a running program[6].

User numbers are displayed in the system prompt, and 32 user areas are supported. The copy utility provides full support between user areas. Also, users logged in as "non-privileged" may not change the current user number by any means.

Other features include attaching the slave console to the master processor, the ability to manually queue files for printing, a rudimentary mailbox facility using FIFO files, a disk verify utility, and automatic reset-and-download for a "crashed" slave.

All of these features will be detailed in the paragraphs to follow.

Documentation

TURBODOS comes with two typeset manuals, a User's Guide and a Configuration Guide. The User's Guide documents all system features and utilities, contrasts the use of TURBODOS with that of CP/M, and contains an extensive theory of operations section. This guide also contains specifications for all of the BDOS system calls (there are about 87 of them, including 47 not available with CP/M).

The Configuration Guide provides complete instructions for using the system generation tools, in addition to providing an informative section describing the modular construction of the operating system. Additionally, the specifications for any user-supplied device drivers are contained here, along with a tutorial on data structures used by TURBODOS drivers, such as linked lists and semaphores. A set of sample driver listings is also included.

The basic manual set was written for revision 1.0 of TURBODOS; the additional system revisions (1.14 is the current version) are documented in the form of supplements to the original manuals. Software 2000 is in the process of re-writing the manuals, and should have them available with the 1.2 release (expected sometime in May).

System Generation

I should preface this discussion by noting that an initial implementation of

(continued next page)

TURBODOS will require CP/M (or some compatible derivative) in order to run the development tools used for system generation. This, of course, does not apply when the hardware and software is purchased from a dealer as a completely integrated package.

TURBODOS is a modularly-constructed operating system with each system function contained within a relocatable module. For example, there are modules for such things as file management (called FILMGR), disk management (a module "closer" to the disk than the file manager, called DSKMGR), a disk buffer manager (BUFMGR), a program loading optimizer (FASLOD), a console manager (CONMGR), and others. These are examples of what Software 2000 calls "kernel-level" modules, meaning they are within the bounds of the operating system. In addition there are "process-level" modules, which are outside of the operating system (meaning that they are concurrent processes, and run simultaneously with other programs, competing for CPU time). Among these are the LCLUSR module (supports a transient program area in a networking master processor), the NETSVC module (handles network I/O requests from slave processors), the DSPOOL module (handles printer despooling). Finally, there are driver-level modules, which are hardware-specific sections, such as CONDR (the lowest-level console I/O handler) and DSKDR (disk driver). These are collectively similar in function to the CP/M BIOS.

The kernel and process level modules are not supplied independently, but are grouped into several different relocatable files, each of which is a functionally distinct version of the operating system. There is a version called STDMASTR, the master networking system; another is called STDSINGL, and is a single-user system; the third is STDSPOOL, a single user version with spooling. There are also groups of modules for slave operating systems, and boot loaders.

The driver level modules exist independently; that is, they are not grouped into a single "library" file, as are the kernel and process level routines, and are supplied also in source form. They can be freely modified by the user and linked into the system at system generation time, allowing complete flexibility in module selection. Since all modules use the Microsoft relocation format, there is a wide selection of assemblers from which to choose.

Curiously, the supplied source modules require the TDL/XITAN series of assemblers (now available from CDL Corp., and Phoenix Software Associates, Ltd.), one of the few relocating assemblers for the Z80 that does NOT produce Microsoft-compatible relocatable modules. For this reason, a utility program is provided (RELCVT.COM) to translate the TDL-format modules to Microsoft format.

Once all the necessary modules are present on the disk, the physical operating system is generated using a program called GEN.COM, which is in fact a linkage editor. The GEN program takes command-line arguments specifying a file that contains the system generation instructions (a ".GEN" file, and optionally a ".PAR" file, which allows modifying individual operating system parameters), the name of the output file (which will contain the configured operating system), and any options, such as "M", which produces a load map, and "S", which produces a symbol table.

GEN.COM must be used to produce at least two files to bring the system up: OSLOADER.COM and OSMASTER.SYS, the operating system loader program, and the master operating system, respectively. Additionally, one slave operating system for each *TYPE* of slave must be present at load time. For example, if all slaves are identical, the file OSSLAVE.SYS should be present on the disk. If different slaves receive different versions of the slave operating system, slave "A" would receive "OSSLAVEA.SYS", slave "B" would receive "OSSLAVEB.SYS", and so on. A table within the master operating system may be modified to specify which slave gets which operating system.

This technique provides for very easy modification and re-configuration of the system, since the included modules can be re-defined by modifying the associated ".GEN" file with a text editor. Operating system parameters, like number of printers (up to 16 printers are supported), buffer-flush delay time, drive step-speed, etc., can be specified by editing the associated ".PAR" file. In fact, the .PAR file can be used to modify *ANY* global symbol in any of the system modules.

Networking

One of the key features of TURBODOS is its networking support. Up to 16 slaves may be associated with one master, allowing full access from each slave to all of the facilities (disk, printers, communications channels, etc.) of the master.

The network interface drivers are supplied with the system, but can be used only with the slave hardware from the dealer. If you want to interconnect the various computers you already have, be ready to write some drivers. This is difficult, because the network requirements are not specified in the manuals (network specifications are promised from Software 2000 for the next release). It should be possible, however, to use the supplied drivers as a guide in writing customized network drivers.

A utility program (MASTER.COM) is provided to allow any slave to attach its console to the master over the network. This allows a rudimentary multiprogramming capability, since it's possible to start a job in the master processor, detach from the master, then run programs simultaneously in the local slave.

Currently, slave-to-slave communication is not supported, nor is master-to-master. Word from Software 2000 is that such communications are planned.

Printer Spooling

TURBODOS supports a very versatile printer spooling/despooling scheme. As many as 16 printers are supported, and flexibility in printer options is provided via three system utilities: PRINT, PRINTER, and QUEUE.

The PRINT utility allows you to control the routing of the printer output (that is, *where* the print characters are sent to). The normal, and most efficient, routing is to a "printer queue", which is a numbered disk file. These files are scheduled for printing on a first-in first-out basis, and are normally deleted at the completion of printing. Queues are designated "A" through "P" and may be associated with a particular printer. For example, the queue

file "-PRINT-A.001" is the name of the first file in the "A" queue, while "-PRINT-B.003" is the third file in the "B" queue. While this may seem confusing, it allows full flexibility in assignment of queues to printers, and permits a distinction between different kinds of output. For example, output requiring a special invoice form may be separated from output requiring a form for shipping, and each may be queued to any printer at a later time.

The PRINT utility also allows print output to be sent directly to any printer (locally, to a printer connected to the slave, or remotely, to one connected to the master), to the console, to a "null" printer (i.e., print output is discarded), or to a disk file (without queueing). The program may also be used to display the current print routing.

The PRINTER utility controls the print destination. This program provides the means of assigning queues to printers, taking printers offline, temporarily halting a print job, or permanently terminating one. For example, the command "PRINT B QUEUE=C" assigns local printer "B" to queue "C".

A good illustration of the use of these utilities is a typical inventory system, where one clerk is handling requisitions, and another inventory reports. Further, let's say there are two printers, "A" (a draft-quality printer, suitable only for rough work) and "B" (a letter-quality printer). Both clerks require letter-quality output for their respective reports, but a secretary is currently using the letter-quality printer for despooling form letters. In this case, the system manager has permanently assigned (via a system generation option) the requisitions clerk (or more accurately, his slave processor) the "A" queue, the inventory reports clerk the "B" queue, and the secretary the "C" queue. Each employee may perform his function, without worrying about what kind of form is currently loaded into the printer, since the output of each is being saved in a disk queue. When the secretary has completed the form letters, the requisitions clerk can then load the requisition forms into the printer, assign the printer to his queue ("PRINT QUEUE=A") and his accumulated output will then be taken from the disk queue and printed out. The clerk doing inventory reports may later on load his report forms into the printer and assign his queue to be printed ("PRINT QUEUE=B"). During this time, any kind of printing may simultaneously take place on the draft printer, with no conflict.

The third utility, called "QUEUE", allows files to be manually marked for despooled printing, and command line options may be used to specify queue assignments, confirmation of individual files (for use with wild-card filename specifications) and automatic deletion after printing. For example, "QUEUE *.PRN ;YDQ=C" will post all of the files with the filetype "PRN" to queue "C"; each file will be presented for confirmation before posting, and each will be deleted after printing.

It should be noted that all of these printer functions are available for programmatic access through BDOS calls, allowing the programmer complete access to any of the spooling and despooling modes.

Communications Channels

TURBODOS supports the concept of communications channels through BDOS calls. These channels may be used for such things as modems, serial ports to various peripherals, or data channels between systems. There may be up to 16 channels in each slave or master. These channels are defined by device drivers contained within the operating system, and are completely definable by the user.

Slaves may access the communications channels of the master over the network. This permits a very convenient means of communication between the master and the slaves.

BDOS calls for use with communications channel access include functions to set baud rate, return channel status, set modem controls, and read modem status.

Batch Processing

TURBODOS comes with a batch processing utility (similar to SUBMIT of CP/M) that allows system commands to be taken from a disk file. This program is called DO.COM, and provides some interesting features.

First of all, command-line substitution arguments are specified within the DO file by number, in a fashion similar to that of CP/M's SUBMIT utility. Unlike SUBMIT, the argument is specified in the DO file by enclosing it in braces. Further, command line arguments may contain embedded spaces, as long as the entire argument is enclosed in quotes.

Another difference between DO.COM and CP/M's SUBMIT.COM is that a temporary disk file is only created if arguments are specified within the DO file. If none are specified, TURBODOS reads the DO file directly. Also, a default argument may be specified within the DO file, for use if no argument is provided within the command line. For example, the DO file line "L80 {1} {2,MYLIB}" defaults the file "MYLIB" if no argument #2 is specified in the DO command line.

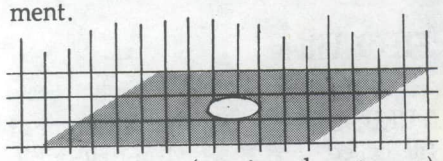
A running DO file supplies input for more than just system commands; in fact, the DO file specifies ALL console input with the exception of direct console I/O (using BDOS function 6). For example, the following is a valid DO file:

```
DDT
D0,7F
↑C
ED FOO.BAR
ICALL FIMP↑ZQ
```

One problem results from this capability: if a running program polls console status to determine if an abort character is waiting at the keyboard, the entire do file will be diverted to the program, since, when console status is found TRUE, the program must consume the waiting character to determine if it is the break character. A notable example of this is the TDL assembler. A possible improvement, therefore, might be the ability to return console status false via a special character within the DO file.

DO files may be nested to any depth[7]; this means that a DO file may contain any number of imbedded DO commands.

DO files may be activated by a user program using a BDOS call, with a pointer to the DO file name as an argument.



(continued next page)

File Interlocks

An important requirement of a multi-user operating system is the need to synchronize access to certain files via a lockout scheme[2]. TURBODOS provides this capability in two ways:

1) Write interlock: Any user writing to a file gains exclusive write-access to that file; attempts by any other user to write to that file returns an error code. This technique provides for one (and only one) "updating" process, with many "inquiry" processes. This means of interlock is automatic, and requires no explicit action on the part of the user or the programmer.

2) Record-level interlock: This feature is applicable to files with the "shared" attribute SET. A process gains a record-level interlock by opening (or creating) the pseudo-file "$.LOK" (which is really not a file at all, but rather a signaling mechanism). Subsequently, any records read by that process become locked, until that record is re-written. Any other process attempting to write a locked record fails in either of two ways:

A. If the conflicting process has not itself opened $.LOK, then it is blocked (removed from the system ready list) until the record becomes available.

B. If the conflicting process has opened $.LOK, then an error message is returned.

Any number of records in any number of files (all must have the shared attribute SET) may be locked concurrently. A process may release all locked records at once by closing $.LOK.

Notably, record-level interlocks may be specified by *any existing* high-level language, providing it has the ability to manipulate disk files. Note that while MP/M II provides for file interlocks, this feature is available only to the assembly language programmer, at least until all systems languages are updated to take advantage of the interlock features.

FIFO Files

TURBODOS provides a special type of file called "FIFO" ("first-in-first-out"). This type of file is useful for communicating between processes and between users, and is created using a utility called FIFO.

The primary significance of a FIFO file is in the way sequential reads and writes act on the file: a record written to a FIFO file is always appended to the end of the file, while a record read is taken from the beginning (and then removed from the FIFO entirely). Random reads and writes work exactly the same with FIFO files as they do with regular files.

A FIFO file may be resident either on disk or in main memory of the master processor. Naturally, the memory-resident FIFO file can be accessed at speeds far in excess of the disk-resident FIFO. The type of the FIFO is specified by answering a query by the FIFO utility. An additional specification to the FIFO utility is the size of the FIFO: disk resident FIFOs may be any size up to the capacity of the disk; memory-resident FIFOs are limited by the available memory space in the master processor.

Two other utility programs provide a rudimentary mail system using FIFO files. The SEND utility posts a message contained in its command line to a specified FIFO file; the RECEIVE program reads a message from a FIFO and displays it on the console.

The SEND utility may be used in combination with the AUTOLOAD and DO facilities to force command lines to idle slave processors. The actual implementation is a little awkward, but does provide an effective means of sending jobs out to slave CPUs as a kind of background processing.

An interesting (and quite useful) feature of FIFOs occurs when the FIFO file has the "shared" file attribute set: if a process attempts to read such a FIFO, and the FIFO is empty, the reading process is blocked until until a record is available at the FIFO. Similarly, a process writing to a full FIFO is blocked until a record is taken (read) from the FIFO. If the "shared" attribute is NOT set, then reading an empty FIFO returns an end-of-file condition; writing to a full FIFO returns a disk full condition.

Applications for FIFOs can best be illustrated by the following example, which is taken from a real situation encountered in my work with TURBODOS.

We have a serial source of input that must occasionally be read by our microcomputer; the device is connected to a spare serial port in one of the slave processors in our system. Normally, files from this source do not exceed 25 or 30 Kbytes, so we've always been able to buffer the incoming data in memory until the input is exhausted, then write the memory buffer to disk. Recently, a customer requested that we process a large job (in excess of 200 Kbytes) with this device.

This presents a problem typical of non-real-time microcomputer operating systems: namely, how to write a buffer to disk without losing incoming serial data. From a slave CPU, disk writes (even though they take place over the network, and are not actually performed by the slave processor) stop the slave processor until the write has actually taken place, and a result (error or no-error) can be returned to the slave processor. Further, at the speed at which our transfer must take place, this required a delay amounting to between 10 and 20 character times per sector (i.e., 10 or 20 characters would be lost each time a sector was flushed to disk).

We found the solution to the problem using a background process in the master processor. This process loops reading the FIFO file "LOGGER.FIL", which is a memory-resident (high-speed) FIFO with the shared attribute set; when there is nothing posted to LOGGER.FIL, the process is blocked from execution. When a record is written to LOGGER.FIL, TURBODOS "wakes up" the background process, and sends it the FIFO record. The first record posted to LOGGER.FIL contains the filename, disk drive, and user number of a to-be-created file where the incoming records will be posted. The background process reads this header record, sets up the requested file, then loops, reading records from LOGGER.FIL; subsequently it posts the records to the requested file, until an end-of-file mark is encountered. At that time it closes the created file, and begins the process all over again.

The slave processor, in the meantime, writes the incoming data record-by-record (after posting the initial header record with filename, disk drive and user number, as specified in the pro-

(continued on page 44)

Bit Manipulation In PL/I-80, Part 2

Michael J. Karas

(*Editor's Note*: The first part of this article described the PL/I-80 bit variable types, showing a program using them.)

The programming example shown below is a means to convert ASCII characters to a corresponding BIT(4) string that "looks" like the hexadecimal form of the original ASCII character. Note that this is the opposite of the function performed by the PUT EDIT output formatting afforded by the B4 format capability. This program uses a built-in function to derive the hex bit pattern for an ASCII character. The INDEX function as:

index(string1,string2)

searches string one from the character 1 position and returns the starting character position where string 2 was found. Zero is returned if string 2 was not found.

```
          hexbit(index(hexseq,desired_char_variable))}


aschex: proc options(main);
        dcl
          /* ascii conversion strings initialized as part of
             .com command file      */

          hexseq char(16) static initial('0123456789ABCDEF'),
          hexbit (16) bit(4) static initial('0'b4,'1'b4,'2'b4,
                                            '3'b4,'4'b4,'5'b4,
                                            '6'b4,'7'b4,'8'b4,
                                            '9'b4,'A'b4,'B'b4,
                                            'C'b4,'D'b4,'E'b4,
                                            'F'b4),
          (i,j)   bin fixed(15),
          bitstr  bit(4),
          txt     file,
          testseq (16) char(1);

         /* setup output file for test data */

          open file(txt) output print title('txt.dat');

         /* build a test pattern string to demo program */

          do i=1 to 15 by 2;   /* plug even values into
                                  front of string       */
          testseq(((i-1)/2) +1)=substr(hexseq,i,1);
          end;
                    .

          do i=2 to 16 by 2;   /* plug odd values into
                                  back of string        */
          testseq(((i-2)/2) +9)=substr(hexseq,i,1);
          end;


    /* the real demo starts here */
          do i=1 to 16;
          bitstr=hexbit(index(hexseq,testseq(i)));
          put file(txt) edit(i,bitstr,bitstr)
                        (col(1),f(2),x(1),b(4),x(1),b4(1));
          end;

        end aschex;
```

The program makes use of this function to determine an integer value for a desired ASCII character to be converted. That is, the sequence '0123456789ABCDEF' has a one-to-one correspondence between the hex/ASCII character and resulting INDEX value for a single character. The INDEX value is used to index an array of the bit(4) values that have bit string patterns in hex equal to the corresponding subscript. Note carefully that the index usage will return a 1 value for an index on the string HEXSEQ if indexed with '0' but at the same time the HEXBIT array subscript contains the corresponding '0000'B string. The expression form shown in this listing returns a bit(4) value equal to the converted hex ASCII character value.

The results of the ASCII to hex conversion are shown here. Each line printed the loop iteration counter followed by the converted bit string value in two formats. The first format is in the binary format to show the value returned from the HEXBIT array indexing algorithm. The second print format gives the printed bit string in hex notation. This convenience is nicely provided by the languages' PUT EDIT capability.

```
 1  0000  0
 2  0010  2
 3  0100  4
 4  0110  6
 5  1000  8
 6  1010  A
 7  1100  C
 8  1110  E
 9  0001  1
10  0011  3
11  0101  5
12  0111  7
13  1001  9
14  1011  B
15  1101  D
16  1111  F
```

This programming example and the one presented last month were provided to allow the reader to become familiar with bit-fiddling in the world of the high level language. However, the language has a few bit manipulation shortcomings. For one thing, the SUBSTR function, if used a lot, is quite slow in operation. Its slowness is due to the general purpose nature of the built-in function. In addition, fast executing functions such as bit string shifting and rotation are not directly available. And finally, if the above example for ASCII to hex conversion seemed cumbersome, then feel right at home with me. That technique is also really slow.

The following assembly language listing is a family of external entry point definitions that provide some nifty functions for bit(16) bit strings. There are functions for rotate and shift "n" places, fast AND, OR, and XOR operations for use on two bit string parameters. And yes!, I've included two simple little function routines to make real quick work of converting ASCII to hex format BIT(4) [actually the first four bits of BIT(16)] and conversion of the upper four bits of BIT(16) to an ASCII character.

The routines are all designed as function call entry points. Parameters are passed by the calling program according to the scheme..

(continued next page)

The [HL] register pair points to a variable address table in memory. The variable address table is a set of two byte memory addresses in a list, one entry for each parameter in the calling statement parameter list. Each list address points to the memory location where the entry variable value is stored. The storage characteristics of each variable are defined by the entry point declarations in the PL/I program. It is up to the assembly language program to fetch the right parameters in the right format for each variable passed.

The assembly routines return their results in the (HL) register pair if the result is a BIT(16) value. In the case of the hex to ASCII function, the result is a character value which gets returned to the PL/I program on the stack.

Definition of the entry points to the assembly language routines are shown below. The appropriate PL/I-80 declare statement to properly define the external assembly language routines as function procedures is given with each example. In all of the examples a PL/I-80 function invocation sequence is given to show how to use the function. The following declare statement is assumed to define the data variables used in the examples:

```
dcl  (bstr1,bstr2)  bit(16),
     i              bin fixed(7),
     cstr           char(1);
```

1) RTROT - Right rotate of bit(16) value. Right rotate entry bit(16) string "n" places where "n" is an entry parameter.

```
dcl   rtrot   entry (bit(16),bin fixed(7))
              returns (bit(16));

bstr1=rtrot(bstr2,3);
```

2) LFROT - Left rotate of bit(16) value. Left rotate entry bit(16) string "n" places where "n" is an entry parameter.

```
dcl  lfrot    entry (bit(16),bin fixed(7))
              returns (bit(16));

bstr2=lfrot(bstr2,8); /* swap byte positions */
```

3) RTSFT - Right shift of bit(16) value. Right logical shift entry bit(16) string "n" places where "n" is an entry parameter. The bit 1 position of the string becomes zero filled.

```
dcl  rtsft    entry (bit(16),bin fixed(7))
              returns (bit(16));

i=6;
bstr1=rtsft(bstr2,i); /* variable shift count */
```

4) LFSFT - Left shift of bit(16) value. Left logical shift of entry bit(16) string "n" places where "n" is an entry parameter. The string becomes zero filled fron the bit 16 position.

```
dcl  lfsft    entry (bit(16),bin fixed(7))
              returns (bit(16));

bstr2=lfsft(bstr2,8);    /* low byte to high byte
                            zero low byte  */
```

5) BITAND - Logical AND of two bit(16) values. Operation is done on a bit by bit basis.

```
dcl  bitand   entry (bit(16),bit(16))
              returns (bit(16));

bstr1=bitand(bstr1,'0000111100000000'b);
```

6) BITOR - Logical OR of two bit(16) values. Operation is done on a bit by bit basis.

```
dcl  bitor    entry (bit(16),bit(16))
              returns (bit(16));

bstr1=bitor(bstr2,'0100'b4);
```

7) BITXOR - Logical XOR of two bit(16) values. The returned value is the bit by bit exclusive or of the operands.

```
dcl  bitxor   entry (bit(16),bit(16))
              returns (bit(16));

bstr2=bitxor('AAAA'b4,bstr1);
```

8) BITASC - Bit to hex/ASCII conversion. Returns an ASCII character that is the ASCII representation of the hex bit pattern in the upper four bits of the entry parameter.

```
dcl  bitasc   entry (bit(16))
              returns (char(1));

cstr=bitasc('1010000000000000'b); /* CSTR = 'A' */
```

9) ASCBIT - Hex/ASCII to bit conversion. The entry ASCII character is converted to the associated hex nibble value in the upper four bits of the returned result.

```
dcl  ascbit   entry (char(1))
              returns (bit(16));

cstr='F';
bstr1=ascbit(cstr);
```

The exact detailed operation of the assembly language is left for the industrious reader to determine. This file on the disk called "PLIBITS.ASM" is made into the proper .REL file with the CP/M system command:

```
A>RMAC PLIBITS<cr>
```

The .REL file is later linked to a PL/I-80 program to include these entry points in the user's program.

```
        title   'pl/i-80 bit manipulation enhancement subroutines'
;*****************************************************************
;
;       enhanced bit processing for pl/i-80 provided by nine
;       assembly language routines. all entry points treated as
;       pl/i-80 function calls with results returned in an immediate
;       fashion to simplify user interface to these routines.
;
;       entry points include:
;               1) Right rotate of bit(16) value
;               2) Left rotate of bit(16) value
;               3) Right shift of bit(16) value
;               4) Left shift of bit(16) value
;               5) Logical AND of two bit(16) values
;               6) Logical OR of two bit(16) values
;               7) Logical XOR of two bit(16) values
;               8) Bit to hex/ASCII conversion
;               9) Hex/ASCII to bit conversion
;
;       written by:
;               Michael J. Karas
;               Micro Resources
;               2468 Hansen Court
;               Simi Valley, California 93065
;               (805) 527-7922
;
;*****************************************************************
```

```
;
; general purpose subroutines used for entry parameter reference
;
;
; get byte entry parameter to register (C)
;
getp1:
        mov     e,m             ;low address of parm
        inx     h
        mov     d,m             ;high address of parm
        inx     h
        xchg                    ;parm ptr to (hl) & current
                                ;..next table ptr to (de)
        mov     c,m             ;single parameter to (c)
        xchg                    ;restore (hl) table ptr for
        ret                     ;..next parameter fetch
;
;
; get word entry parameter to (BC)
;
getp2:
        call    getp1           ;get low parameter byte in (C)
        inx     d               ;bump parameter pointer
                                ;..still valid from getpl
        ldax    d               ;get high parm byte value
        mov     b,a             ;put high to (B)
        ret
;
;
; fetch parameter sequence bit(16) and bin fixed(7) masked to
; four places to (DE) and (B) respectively
;
gbitcnt:
        call    getp2           ;fetch our bit string
        push    b               ;save
        call    getp1           ;get count
        mvi     a,0fh           ;mask to 16 counts
        ana     c
        mov     b,a             ;count to (B)
        pop     d               ;bit string to (DE)
        ret
;
;
; fetch parameter sequence bit(16) and bit(16) to
; (DE) and (BC) respectively
;
gbitbit:
        call    getp2           ;fetch first bit string
        push    b               ;save
        call    getp2           ;get second bit value
        pop     d               ;first parm to (BC)
        ret
;
;
;*****************************************************************
;
; function number 1: right rotate bit(16) value n places
;
        public  rtrot
;
; entry,
;       bit(16) value string to be right rotated
;       bin fixed(7) value number of places to right
; returns,
;       right rotated bit(16) value as a function call
;
rtrot:
        call    gbitcnt         ;fetch parameters
rtrotl:
        mov     a,e             ;get lsb of low to carry
        rrc
        mov     a,d             ;rotate top byte
        rar                     ;..lsb of D to carry
        mov     d,a             ;put high byte back
        mov     a,e             ;rotate low byte
        rar                     ;..moving carry in
        mov     e,a
        dcr     b               ;dec rotate count
        jnz     rtrotl          ;? more to do
        xchg                    ;result to (hl) for return
        ret
;
;
;*****************************************************************
;
; function number 2: left rotate bit(16) value n places
;
        public  lfrot
;
; entry,
;       bit(16) value to be left rotated
;       bin fixed(7) value number of places to left
; returns,
;       left rotated bit(16) value as a function call
;
lfrot:
        call    gbitcnt         ;fetch parameters
lfrotl:
        mov     a,e             ;get msb of low to carry
        rlc
        mov     a,d             ;rotate top byte
        ral                     ;..msb of D to carry
        mov     d,a             ;put high byte back
        mov     a,e             ;rotate low byte
        ral                     ;..moving carry in
        mov     e,a
        dcr     b               ;dec rotate count
        jnz     lfrotl          ;? more to do
        xchg                    ;result to (hl) for return
        ret
;
;
;*****************************************************************
;
; function number 3: right shift bit(16) value n places
;
        public  rtsft
;
; entry,
;       bit(16) value string to be right shifted
;       bin fixed(7) value number of places to right
; returns,
;       right shifted bit(16) value as a function call
;
rtsft:
        call    gbitcnt         ;fetch parameters
rtshtl:
        ora     a               ;clear carry for zero fill
        mov     a,d             ;shift top byte
        rar                     ;..lsb of D to carry
        mov     d,a             ;put high byte back
        mov     a,e             ;shift low byte
        rar                     ;..moving carry in
        mov     e,a
        dcr     b               ;dec shift count
        jnz     rtshtl          ;? more to do
        xchg                    ;result to (hl) for return
        ret
;
;
;*****************************************************************
;
; function number 4: left shift bit(16) value n places
;
        public  lfsft
;
; entry,
;       bit(16) value to be left shifted
;       bin fixed(7) value number of places to left
; returns,
;       left shifted bit(16) value as a function call
;
lfsft:
        call    gbitcnt         ;fetch parameters
lfshtl:
        ora     a               ;clear carry for zero fill
        mov     a,e             ;shift low byte
        ral                     ;..msb of E to carry
        mov     e,a             ;put low byte back
        mov     a,d             ;shift high byte
        ral                     ;..moving carry in
        mov     d,a
        dcr     b               ;dec rotate count
        jnz     lfshtl          ;? more to do
        xchg                    ;result to (hl) for return
        ret
;
;
;*****************************************************************
;
; function number 5: bitwise and two bit(16) values
;
        public  bitand
;
; entry,
;       two bit(16) values to logically AND
; returns,
;       result as bit(16) value like a function call
;
bitand:
        call    gbitbit         ;fetch two parms
        mov     a,b             ;do top two bytes
        ana     d
        mov     h,a
        mov     a,c             ;now low two bytes
        ana     e
        mov     l,a
        ret
;
;
;*****************************************************************
;
; function number 6: bitwise or two bit(16) values
;
        public  bitor
;
; entry,
;       two bit(16) values to logically OR
; returns,
;       result as bit(16) value like a function call
;
bitor:
        call    gbitbit         ;fetch two parms
        mov     a,b             ;do top two bytes
        ora     d
        mov     h,a
        mov     a,c             ;now low two bytes
        ora     e
        mov     l,a
        ret
;
;
;*****************************************************************
;
; function number 7: bitwise exclusive or of two bit(16) values
;
        public  bitxor
;
; entry,
;       two bit(16) values to logically XOR
; returns,
;       result as bit(16) value like a function call
;
bitxor:
        call    gbitbit         ;fetch two parms
        mov     a,b             ;do top two bytes
        xra     d
        mov     h,a
        mov     a,c             ;now low two bytes
        xra     e
        mov     l,a
        ret
;
;
;*****************************************************************
```

(continued next page)

```
;
; function number 8: convert bit value to hex ascii
;
        public  bitasc
;
; entry,
;       bit(16) value whose upper four bits
;       are hex nibble converted to a hex ascii representation
; returns,
;       resulting char(1) returned as a function call
;
bitasc:
        call    getp2           ;get pit string to (BC)
        mov     a,b             ;position high nibble
        rrc
        rrc
        rrc
        rrc
        ani     0fh             ;low nibble only
        adi     090h            ;use famous 6 byte conversion
        daa
        aci     040h
        daa
        pop     h               ;get pli/80 return address
        push    psw             ;char to stack
        inx     sp              ;ignore flags
        mvi     a,1             ;one char size
        push    h               ;return address back to stack
        ret
;
;
;*****************************************************************
;
; function number 9: convert hex ascii to bit value
;
        public  ascbit
;
; entry,
;       char(1) ascii character value that is
;       converted to a bit(16) value with the upper four
;       bits corresponding to the hex/ascii character
; returns,
;       bit(16) value as a function call
;
ascbit:
        call    getpl           ;get character to (C)
        mov     a,c
        sui     '0'             ;base at zero
        cpi     010             ;need offset adjust?
        jc      ascl
        sui     07h
ascl:
        ani     0fh             ;mask to four bits
        rlc                     ;move to upper nibble
        rlc
        rlc
        rlc
        mov     h,a
        mvi     l,0             ;zero lower bits
        ret
;
        end
;
;
;+++...end of file plibits.asm
```

The entry point definition given above showed an example declare statement for each assembly language routine. The following small file called "PLIBITS.DCL" is a one statement declare file that is put into the users PL/I-80 program with a statement like:

```
%include 'PLIBITS.DCL';
```

This prevents the programmer from having to retype the declared entry point definition each time the functions of PLIBITS is desired. It also saves mistakes.

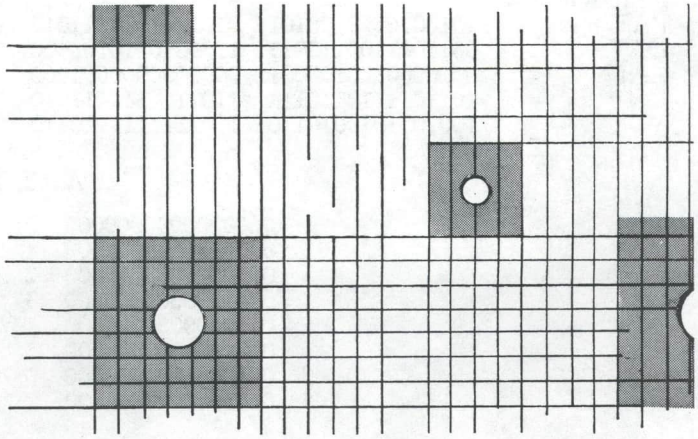
```
/* enhanced bit operation performance subroutine
   entry point definition                          */
dcl
        rtrot   entry (bit(16),bin fixed(7))
                returns (bit(16)),
        lfrot   entry (bit(16),bin fixed(7))
                returns (bit(16)),
        rtsft   entry (bit(16),bin fixed(7))
                returns (bit(16)),
        lfsft   entry (bit(16),bin fixed(7))
                returns (bit(16)),
        bitand  entry (bit(16),bit(16))
                returns (bit(16)),
        bitor   entry (bit(16),bit(16))
                returns (bit(16)),
        bitxor  entry (bit(16),bit(16))
                returns (bit(16)),
        bitasc  entry (bit(16))
                 returns (char(1)),
        ascbit  entry (char(1))
                returns (bit(16));
```

A comprehensive test program to show operation of all the assembly language routines is given below. Once again the program structure and operation is left for the reader to tear apart. The listing output from this program is given at the end of the article. To compile and link the program source contained in "BITTST.PLI" use the commands:

```
A>PLI BITTST<cr>
A>LINK BITTST,PLIBITS<cr>
```

```
bittst: proc options(main);

%include 'plibits.dcl'; /* bring in the bit entry point dcl */

        dcl
                tb (5)  bit(16) static init('a5a5'b4,
                                            '5a5a'b4,
                                            '1234'b4,
                                            'fedc'b4,
                                            '0ace'b4),
                pb (5)  bit(16) static init('3333'b4,
                                            '85ac'b4,
                                            'ffff'b4,
                                            '0000'b4,
                                            'f0a5'b4),
                (i,j,k) bin fixed(7),
                (bita,bitb) bit(16),
                string char(16) static init('0123456789ABCDEF'),
                syslst  file;

        open file(syslst) output print linesize(132)
                pagesize(57) title('$lst');

        do i=1 to 5;
        put file(syslst) edit(tb(i))(col(1),b);
        end;

        put file(syslst) skip(4);

        do j=1 repeat(j*2) while(j <= 16);
        put file(syslst) skip(1) edit('Count Value = ',j)
                                (col(1),a,f(2));

                put file(syslst) edit('PATTERN')(col(1),a);
                do i=1 to 5;
                put file(syslst) edit(tb(i))(x(2),b(16));
                end;

                put file(syslst) edit('RT ROT ')(col(1),a);
                do i=1 to 5;
                put file(syslst) edit(rtrot(tb(i),j))(x(2),b(16)
                end;

                put file(syslst) edit('LF ROT ')(col(1),a);
                do i=1 to 5;
                put file(syslst) edit(lfrot(tb(i),j))(x(2),b(16))
                end;

                put file(syslst) edit('RT SFT ')(col(1),a);
                do i=1 to 5;
                put file(syslst) edit(rtsft(tb(i),j))(x(2),b(16))
                end;

                put file(syslst) edit('LF SFT ')(col(1),a);
                do i=1 to 5;
                put file(syslst) edit(lfsft(tb(i),j))(x(2),b(16))
                end;

        end;

        put file(syslst) skip(4);

        do j=1 to 5;
        put file(syslst) skip(4) edit('Pattern number ',j)
                                (col(1),a,f(1));

                put file(syslst) skip edit('PAT ')(a);

        /* pattern */
                do i=1 to 5;
                put file(syslst) edit(tb(i))(x(2),b(16));
                end;

                put file(syslst) skip edit('DATA')(a);

        /* test value */
                do i=1 to 5;
                put file(syslst) edit(pb(i))(x(2),b(16));
                end;

        /* AND */
                put file(syslst) skip edit('AND ')(a);

                do i=1 to 5;
                put file(syslst) edit(bitand(tb(i),pb(i)))
                                (x(2),b(16));
                end;

        /* OR */
                put file(syslst) skip edit('OR  ')(a);

                do i=1 to 5;
                put file(syslst) edit(bitor(tb(i),pb(i)))
                                (x(2),b(16));
                end;

        /* XOR */
                put file(syslst) skip edit('XOR ')(a);

                do i=1 to 5;
                put file(syslst) edit(bitxor(tb(i),pb(i)))
                                (x(2),b(16));
                end;

        end;

        put file(syslst) skip(4) edit('ASCII Translation')
                                (col(1),a);

        do j=1 to 16;
```

```
put file(syslst) edit(j,ascbit(substr(string,j,1)),
                 bitasc(ascbit(substr(string,j,1))))
                 (col(1),f(2),x(2),b(16),x(1),a);
end;

end bittst;
```



The output from the above shown program is given below. All functions of the assembly language routines are exercised.

```
1010010110100101
0101101001011010
0001001000110100
1111111011011100
0000101011001110
```

```
Count Value =1
PATTERN     010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
RT ROT      101001011010010  010110100101101  000100100011010  111111101101110  000010101100111
LF ROT      100101101001011  011010010110100  010010001101000  111110110111001  001010110011100
RT SFT      101001011010010  010110100101101  000100100011010  111111101101110  000010101100111
LF SFT      100101101001010  011010010110100  010010001101000  111110110111000  001010110011100
Count Value = 2
PATTERN     010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
RT ROT      110100101101001  001011010010110  000010010001101  011111110110111  000001010110011
LF ROT      001011010010110  110100101101001  100100011010000  111101101110011  010101100111000
RT SFT      010100101101001  001011010010110  000010010001101  011111110110111  000001010110011
LF SFT      001011010010100  110100101101000  100100011010000  111101101110000  010101100111000
Count Value = 4
PATTERN     010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
RT ROT      101101001011010  010010110100101  100000100100011  100111111101101  110000010101100
LF ROT      101101001011010  010010110100101  010001101000001  110110111001111  010110011100000
RT SFT      000101001011010  000010110100101  000000100100011  000111111101101  000000010101100
LF SFT      101101001010000  010010110100000  010001101000000  110110111000000  010110011100000
Count Value = 8
PATTERN     010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
RT ROT      010010110100101  101101001011010  011010000010010  101110011111110  100111000001010
LF ROT      010010110100101  101101001011010  011010000010010  101110011111110  100111000001010
RT SFT      000000010100101  000000001011010  000000000010010  000000011111110  000000000001010
LF SFT      010010100000000  101101000000000  011010000000000  101110000000000  100111000000000
Count Value = 16
PATTERN     010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
RT ROT      010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
LF ROT      010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
RT SFT      000000000000000  000000000000000  000000000000000  000000000000000  000000000000000
LF SFT      000000000000000  000000000000000  000000000000000  000000000000000  000000000000000

Pattern number 1
PAT         010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
DATA        011001100110011  000010110101100  111111111111111  000000000000000  111000010100101
AND         010000100100001  000000000001000  001001000110100  000000000000000  000000010000100
OR          011011110110111  101111111111110  111111111111111  111111011011100  111101011101111
XOR         001011010010110  101111111110110  110110111001011  111111011011100  111101001101011

Pattern number 2
PAT         010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
DATA        011001100110011  000010110101100  111111111111111  000000000000000  111000010100101
AND         010000100100001  000000000001000  001001000110100  000000000000000  000000010000100
OR          011011110110111  101111111111110  111111111111111  111111011011100  111101011101111
XOR         001011010010110  101111111110110  110110111001011  111111011011100  111101001101011

Pattern number 3
PAT         010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
DATA        011001100110011  000010110101100  111111111111111  000000000000000  111000010100101
AND         010000100100001  000000000001000  001001000110100  000000000000000  000000010000100
OR          011011110110111  101111111111110  111111111111111  111111011011100  111101011101111
XOR         001011010010110  101111111110110  110110111001011  111111011011100  111101001101011

Pattern number 4
PAT         010010110100101  101101001011010  001001000110100  111111011011100  000101011001110
DATA        011001100110011  000010110101100  111111111111111  000000000000000  111000010100101
AND         0100001001       0000000000001000 0001001000110100 0000000000000000 0000000010000100
OR          1011011110110111 1101111111111110 1111111111111111 1111111011011100 1111101011101111
XOR         1001011010010110 1101111111110110 1110110111001011 1111111011011100 1111101001101011
```

(continued next page)

Pattern number 5

PAT	1010010110100101	0101101001011010	0001001000110100	1111111011011100	0000101011001110
DATA	0011001100110011	1000010110101100	1111111111111111	0000000000000000	1111000010100101
AND	0010000100100001	0000000000001000	0001001000110100	0000000000000000	0000000010000100
OR	1011011110110111	1101111111111110	1111111111111111	1111111011011100	1111101011101111
XOR	1001011010010110	1101111111110110	1110110111001011	1111111011011100	1111101001101011

ASCII Translation

1	0000000000000000	0	9	1000000000000000	8
2	0001000000000000	1	10	1001000000000000	9
3	0010000000000000	2	11	1010000000000000	A
4	0011000000000000	3	12	1011000000000000	B
5	0100000000000000	4	13	1100000000000000	C
6	0101000000000000	5	14	1101000000000000	D
7	0110000000000000	6	15	1110000000000000	E
8	0111000000000000	7	16	1111000000000000	F

Volume 80 and Abstracts

CP/M Users Group

Volume 80

DESCRIPTION: Cromemco Structured BASIC programs by David E. Trachtenbarg:

1. Mail list programs.
2. Spelling programs.
3. Statistical programs.
4. Misc. other programs.

NO.	SIZE	NAME	COMMENTS
		CATALOG.080	CONTENTS OF CP/M VOL. 080
		ABSTRACT.080	Volume abstract
	1K	FILES.CRC	CRC of files on disk
	2K	CRCK.COM	CRC program.
	5K	U-G-FORM.LIB	CPMUG submission form.
80.1	2K	BACKUP.STB	Part of mail list system
80.2	6K	CEDIT.STB	Part of spelling program
80.3	2K	CHECK.DAT	Part of spelling program
80.4	8K	CONV-ASC.STB	Convert MBASIC to Cromemco
80.5	6K	CONV-BAS.STB	structured BASIC
80.6	2K	DATE.STB	Part of mail list system
80.7	10K	DEDIT.STB	Part of spelling program
80.8	2K	DICTION.DAT	Part of spelling program
80.9	12K	GRADER.STB	Text evaluation program
80.10	4K	MMENU.STB	Part of mail list system
80.11	2K	PRN-TEST.STB	Printer test program
80.12	10K	REC-EDIT.STB	Part of mail list system
80.13	8K	REC-PRN.STB	Part of mail list system
80.14	2K	SMENU.STB	Part of spelling program
80.15	8K	SORTS.STB	Sort routines

The programs on volume 80 were contributed by David E. Trachtenbarg of Peoria, IL.

Here are a few comments extracted from his very complete submittal forms:

* They all require Cromemco 32K Structured BASIC

* Most have hardware dependency on a Hazeltine 1500 terminal. The statistical programs have optional plotting routines that require a dot matrix printer.

Ward Christensen

This disk contains a set of programs that I wish to donate to CPMUG for noncommercial use. Most are entirely original programs that I wrote from scratch. Others are based on algorithms that described by others. None have been published as Structured BASIC programs before to my knowledge.

Although all of the programs work, some still have minor bugs in them that I know about. All of the programs could be improved upon.

SMENU.STB, SPELL.STB, CEDIT.STB, DEDIT.STB, CHECK.DAT, TRANSFER.STB, DICTION.DAT and WORDLIST. TXT are all parts of a spelling checking program. The program was written before the commercial spelling checkers were first marketed. Unlike the commercial checkers it is very slow, since each word is checked in a ISAM dictionary file.

A text file containing a short list of common words is also included. The program will scan a text file and place all of the words not in the dictionary in another file to be checked. After all of the words in the check file have been reviewed they may be added to the dictionary. The writer must use an

(continued on page 43)

Developing Applications With dBASE II

Steve Patchen

We are entering an era in which new forms of program creation are emerging. Most existing application programs were created by programmers writing code one line at a time. This is a slow and costly way to create data processing applications. Alternative strategies with such names as "application generators" and "program building systems" are evolving. Associated with these new techniques are database systems and fourth generation languages. These new tools provide direct access to data and allow many useful manipulations of data without any programming. James Martin's latest book "Application Development Without Programmers", published by Prentice-Hall, provides a good introduction to some techniques currently in use. The National CSS 'NOMAD' system (Wilton,CT, 1981) discussed in his book is a relational database system similar to dBASE II. However, it has a better report facility and other greater powers, partly because it has a more complex data definition strategy and also because it was designed for large capacity mainframe computers.

These tools alone, however, do not solve all the difficulties which arise in data processing. Some problems become critical as more applications are added to a data processing site, such as the loss of the ability to keep track of the current state of data in the system and the difficulty in making data created by one application available to another. A tool called a data dictionary is available on many mainframe computers to ease these problems. A data dictionary could also be capable of keeping track of other objects in the system, like program revisions and report formats. Data dictionaries and database systems complement each other. The database facilitates accessing data as a separate entity from programs and the dictionary keeps track of the contents and structures of the data processing system. These tools are natural prerequisites to any application creation system.

For the past year I have been working on an application development system which includes a data dictionary, a project management facility and an application build system. The current version of this system uses dBASE II version 2.03 as a foundation. The data dictionary is used to contain the application design; it also provides supplementary data definitions for screen and report formats and for complex data sets. The build system includes a menu facility. This menu system is an essential structure in both the development system and in applications generated by it.

Menu systems are commonly used in application software. They are usually restricted in structure to suit the particular application. Their facilities seldom extend to operating system utilities or include the ability to handle other modes of interaction with the user of the system. There is no reason why a menu system cannot be used as an integral part of the operating system to provide easy access to the computer for new and inexperienced users, and still allow lower levels of operation by the more experienced operator. The same menu control module is used by both the development system and the application. Directories seldom provide adequate information to users of the system. Tying the parts of the system together with a menu insures that users have access to the facilities they need. Data dictionaries can then provide additional information for system maintenance.

The program which controls the menu is generalized and data driven. That is, there is only one menu program, but there can be as many menus as required. The menus are stored in a data file which can be easily accessed to make additions and changes. The menu control program and the menu editor show good examples of the ease and convenience in many of the higher level commands provided by a new generation of software. dBASE II is versatile enough to provide low level support for a wide range of applications. Thus it is possible to build modules for applications suited to general use and still have the ability to customize modules in the field for special requirements. This represents a compromise between unmodifiable packaged software and expensive custom written programming. I have selected the menu system for more detailed discussion as a good example of the structure of an application developed for dBASE II. The rest of this article will be devoted to a discussion of the menu system and to the strategies used in taking advantage of dBASE II's fourth generation features; the strategies used to build modules designed for construction of applications will also be considered.

Note that each of the programs and data structures have a copyright. You may copy the programs for your own use but please do not copy them in any form for trade or money. I have also reserved the trademark of DATAWARE for data driven software created by myself at Dataware Systems. Please do not remove the copyrights from your own copies.

First let's look at how the menu system works and how it can be used. Then I will discuss the menu control program, the menu editor and report routines in more detail along with some of the dBASE II commands they use. Figure 1 is an example run of the system. The menu control program is started either with a 'DBASE MENU' command from CP/M-80 or with a 'DO MENU' command from dBASE II. The menu displayed is the last one in use unless the system is being run for the first time. The last date entered in the system is retained and an opportunity to change it is offered each time the menu system is entered. After the current menu is displayed only the characters shown as selections are accepted as input. Any other entry produces an error message. The bottom four selections are displayed with every menu. They are built into the menu control program to allow interaction outside the normal menu routing. The '>' selection allows any CP/M-80 command to be entered; upon completion the menu is restored. The '.' selection similarly allows a one line dBASE II command to be executed. The syntax of the command line entered is checked with the new TEST() function to avoid bombing out from an entry error. The '*' selection allows you to bypass the normal routing through the

(continued next page)

menus by entering the two character name of the menu you wish. The last built-in selection is the '?' character. It causes a help message for an indicated selection to be displayed.

The other selections in any menu are listed directly from a data file each time the menu is displayed. The selection character can be any displayable character, but the menu editor automatically loads numbers corresponding to the selection position when the menu is first created. If you use one of the four built-in characters the selection will never be accessed because the built-in functions take precedence. The next column of the menu display contains a short forty-character description of the selection, but a better description should be placed in the help entry for each selection. The last column displayed is either the menu or function name for the selection, depending upon whether it is a menu selection or a program function. Ten characters are allowed for the function name to accommodate device designations as well as eight character CP/M-80 names. The CP/M-80 extension for functions is always '.CMD' for dBASE II command files and thus is not shown. The example run shows a routing into the build system menu and then into the menu system maintenance routines.

I used the SET ALTERNATE TO filename and the SET ALTERNATE ON commands to generate the trial run in figure 1. The menu editor does not show the editing screen displays because, for some reason, screen format displays are not echoed into the file created. The screen format files for the editor are listed with the other source files in figure 2. To clarify things not revealed by the echo of the run and to economize on space, I have edited the dump a little. Changes and notes are in parentheses. After doing a help display I routed through menus to the menu editor. The menu editor shows you which files are being edited and then asks for the name of the menu to edit. It tries to locate the menu and display it for confirmation. If it cannot find the menu requested you are offered an opportunity to create it. You are asked to enter the number of entries desired in this new menu; then the program creates it and prompts for you to fill in the menu entries, using the MENU.FMT screen entry form. When an 'R' or an 'E' is entered for a command letter the second screen format is used to fill in the report or screen entry parameters for this selection. After leaving, the screen entry displays a null entry and backs up one step in the program; so in order to exit you just hit RETURN keys until you are back in the menu. I left the menu by giving a direct CP/M-80 command to run WordStar. The shortest route back to the main entry menu is to use the '*' selection and enter a null to get the default menu (the first one in the file).

For the following detailed discussion refer to the source listings in figure 2. The menu control module first disables the ESCAPE key and the interactive talk. It also makes sure that the screen will be the display destination. The second section of code is used to insure that the memory variables used by the control module are loaded the first time the menu system is run or when changing to another menu system. You can change to another menu system either by deleting the MENU.MEM file or by calling a routine which changes the menu parameters and saves them to MENU.MEM. The dBASE II commands SAVE and RESTORE are used to save the memory variables anytime they are changed and to restore the system to the menu it had left after doing some function. The next section of code allows the user to correct the date if it is required.

The rest of the routine is an infinite do while loop. Most returns from dBASE II functions remain within this loop. The top of the loop consists of the menu display. It is in three parts: the first part is just the menu title; the second part is displayed directly from the current menu file; the last part consists of the built-in functions and the column headings. The last line is a prompt to enter a selection. The WAIT command displays 'WAITING' on the screen and accepts one character from the operator. The remainder of the do while loop is a response to the character entered.

The entered character is first checked against the four built-in selections. If is not one of these, the current menu records are searched for a selection match. An error message is displayed if it is not found. Otherwise, the action taken depends upon the content of the COMMAND and the FUNCTION fields in the record located. The command letter "C" causes the FUNCTION field contents to be used as a CP/M-80 command. "D" uses the description. A command letter of 'M' causes the first two characters of the FUNCTION field to be loaded as the current menu name. This then becomes the next display seen. If the COMMAND letter is an 'F' the FUNCTION field contents are executed as a dBASE II command. If it is either an 'E' or an 'R' the FUNCTION field from the menu file is used as a key to locate the record used to get information for entry and report routines. The function found in this file is then executed as a program. All routines called return to the menu control module and redisplay the previous menu.

The MENUEDIT.CMD takes advantage of the REMARK command to provide both the program title and the introductory screen display. The parameters required by the routine are checked to insure that they are not null. Then the files are selected, the operator is informed of which files are being edited and asked which menu he or she wishes to edit. The program enters a do while loop and does not leave it until the operator enters a null response to the prompt for a new menu to edit at the bottom of the loop. If the menu requested is already in the file the FIND command will locate it; if not the EOF function will be set true. This causes the ELSE part of the IF statement, which asks the operator if he wishes to create the specified menu, to be executed.

The menus are printed using two generalized routines and one customized for the menus. The report routines use most of the parameters loaded by the menu system from the report format file. The form length which I am currently using is for 8 ½" by 14" paper; I cut it down to 8 1/2" by 11" for binders without breaking the form seams. If you use different paper just employ a different form length in the MENURPT.DBF file record for this report. REPORT1.CMD is called by the menu after loading the parameters. To avoid line wraparound on 80 column printers the 132 character RHEADING string is trimmed down to only the characters you have entered into it. The menu files are selected, and then the printing takes place within three nested do while loops. The first loop continues the printing until the end of the primary file is reached. The second loop makes a paging decision following the records referenced by the KEYA parameter. The key group title line is printed and the third loop is entered to print the associated records. KEYB is used as the secondary key to select possible multiple records in this loop. PFORMAT contains the name of the routine to use to print these records. When returning from this routine another paging decision is made. The PFORMAT routine in our case is MENURPT1.-

CMD. The routine TOPOFPAG.CMD is called to handle the paging and top of page heading. It trims the '____' line to match the RHEADER line length before printing it. Before leaving the routine all the memory variables not needed any more are released and the paper is ejected out of the printer. Figure 3 is an example of this report.

REPORT1.CMD is generalized to handle a wide class of reports which print columns of entries. The routine loaded into PFORMAT can be either a customized routine or another generalized routine to handle a class of column reports. A simple such generalized routine would be:

```
SELECT SECOND
LIST OFF FOR &KEYB$KEYBVAL
SELECT PRIMARY
```

A composite expression of KEYA and KEYB might also be used. The DISPLAY or LIST command can be formatted as in the menu display routine. Another form of report (as for invoices and purchase orders) would use a file containing the location of fields to print on preprinted forms. The file would contain a format key, a series of field names and a line and column position for each field. These field entries would be sorted on the format key, on the line and on the column to make sure they are printed in the proper order. The report routine would read through this series of records for each form printed, placing the current values for each field in correct locations with the '@ line,column SAY &field' command. If two files are required for the form, as in multiple item entries, the second file would require another set of format records for the line items. Only the column position would be used from the file because the item would be printed in the current item row. The FIND, LOCATE, LIST and DISPLAY commands are examples of the difference in power between dBASE II and similar databases and the cumbersome methods of performing these common operations in BASIC, FORTRAN or other older languages. The FIND command is used with an index while the LOCATE command allows complex expressions specifying which file record to retrieve. The macro command '&' is used heavily in these programs; this feature allows programs to be data driven. Any part of a command line can be formed as data and substituted into a program during execution, so that direct CP/M-80 and dBASE II command selections in the menu are possible, as is building applications from data specifications. Thus routines like REPORT1.CMD can be created and used by many reports. Or molds for programs can be filled in from data files and .CMD files created with the COPY SDF command.

Keep in mind that databases and data driven software make heavy use of the mass storage media. Thus the access time limitations of your mass storage can have a significant effect upon the performance of the system. To give you some idea, I compared some times running the menu system from both my 8" floppy system and my winchester. The average time to change menus on the 8" was 12 seconds but on the winchester was less then 8 seconds. The menu editor loaded in 9 seconds from the floppies and in only 3 seconds from the hard disk. The return to the menu took 10 seconds and 6 seconds respectively. My computer has a 4MHZ Z80 processor. If you have a 2MHZ 8080 or 5¼" drives the response times will be slower. The menu display can be speeded up a little by removing unused help message fields if desired. I recommend that you develop the application first and then examine the file to see if the longest message uses all the fields. If it doesn't you can make a copy of the structure and modify it. Just append from the original file and rename it. Then don't forget to remove the appropriate ? L7 etc. from the MENU.CMD file. Menus imbedded in programs will run faster then these menus, but modifications to the menus will require a programmer and program changes with associated debugging. This menu system can be maintained by the user of the system.

I have not provided full error trapping in these routines because dBASE II does not have built-in error trapping facilities and doing the error coding in line with the functions obscures the structure of the routines. There is some error trapping to provide reasonable operation convenience. The development system of which this is a part provides for both manual and automated application generation features. As enough of the system is completed and tested I will provide further examples of the structure of the system. The parts needed to complete a minimum system are the build routines to do file creation, screen formats and report formats.

For those of you who do not wish to type the programs and structures into dBASE II I will provide copies on 8" single density diskettes for $45 each. I will include the latest version of the routines and documentation on the diskette. My ability to deliver copies is limited, so please only order it if you intend to experiment with these routines and provide some feedback from your experience with them. Send pre-paid orders to Dataware Systems, 255 Chippewa, Pontiac, Mich. 48053. Allow 30 days. I will send an acknowledgement of orders received.

Figure 1 A Menu Test Run

```
A>DBASE MENU

 DATAWARE (TM) MENU SYSTEM COPYRIGHT 1982 BY STEVE PATCHEN

              CORRECT THE DATE IF REQUIRED 03/08/82

(SCREEN IS ERASED)

      ********************************************************************************
      *    03/02/82          DATAWARE SYSTEMS  (TM)                                  *
      *------------------------------------------------------------------------------*
      *                 THE DATAWARE MANAGEMENT SYSTEM  (TM)                         *
      *                                                                              *
      *    1       The DATA DICTIONARY                          B1                   *
      *    2       The PROJECT CHART                            B2                   *
      *    3       The BUILD SYSTEM                             B3                   *
      *    4       Project initialization                       B4                   *
      *    5       SYSTEM HOUSEKEEPING ROUTINES                 B5                   *
      *                                                                              *
      *    >       To do a CP/M command                                              *
      *    .       To do a DBASE II command                                          *
      *    *       To go to a MENU not listed by entering its name                   *
      *    ?       To see help messages for any selection                            *
      *------------------------------------------------------------------------------*
      * SELECTION        DESCRIPTION                  MENU/FUNCTION NAME             *
      ********************************************************************************
                        ENTER A SELECTION CHARACTER!
      WAITING ?
      ENTER the SELECTION character to get help for. :?

       THE ? SELECTION IS USED TO VIEW HELP MESSAGES FOR ANY SELECTION IN THE MENU.
      Respond to the prompt with the character for the selection you wish to view.


       ENTER any key to continue.
      WAITING

( THE ORIGINAL MENU IS REDISPLAYED HERE)

                        ENTER A SELECTION CHARACTER!
      WAITING 3

(SCREEN IS ERASED)

      ********************************************************************************
      *    1/02/82           DATAWARE SYSTEMS  (TM)                                  *
      *------------------------------------------------------------------------------*
      *                          THE BUILD SYSTEM                                    *
      *                                                                              *
      *    1       Build or Edit the Menus                      M0                   *
      *    2       Build data files                             N0                   *
      *    3       Build Reports                                R0                   *
      *    4       Build entry screens                          S0                   *
      *    5       Build Program modules                        P0                   *
      *    6       RETURN TO THE MAINMENU                       A0                   *
      *                                                                              *
      *    >       To do a CP/M command                                              *
      *    .       To do a DBASE II command                                          *
      *    *       To go to a MENU not listed by entering its name                   *
      *    ?       To see help messages for any selection                            *
      *------------------------------------------------------------------------------*
      * SELECTION        DESCRIPTION                  MENU/FUNCTION NAME             *
      ********************************************************************************
                        ENTER A SELECTION CHARACTER!
```

(continued next page)

```
      WAITING 1

(SCREEN IS ERASED)

      ********************************************************************************
      *     03/02/82          DATAWARE SYSTEMS  (TM)                                 *
      *------------------------------------------------------------------------------*
      *                       BUILD OR EDIT THE MENUS                                *
      *                                                                              *
      *    1       Build menus from the dictionary          GENMENUS                 *
      *    2       Add or edit menus                        MENU                     *
      *    3       Delete a program call from all menus     DELFUNCT                 *
      *    4       Add a program call to selected menus     ADDFUNCT                 *
      *    5       Print the menus                          PRINTMEN                 *
      *    6       Print the help messages                  PRINTHLP                 *
      *    7       RETURN TO THE BUILD MENU                 B3                       *
      *                                                                              *
      *    >       To do a CP/M command                                              *
      *    .       To do a DBASE II command                                          *
      *    *       To go to a MENU not listed by entering its name                   *
      *    ?       To see help messages for any selection                            *
      *------------------------------------------------------------------------------*
      *  SELECTION         DESCRIPTION                 MENU/FUNCTION NAME           *
      ********************************************************************************
                        ENTER A SELECTION CHARACTER!
      WAITING 2

(SCREEN IS ERASED)
       EDITMENU.CMD   COPYRIGHT 1982 BY STEVE PATCHEN
        THIS ROUTINE ALLOWS EDITING OF MENU FILES
     The MENU       AND MENURPT     are being edited.
     WHICH MENU DO YOU WISH TO ADD TO, DELETE FROM OR OTHERWISE CHANGE. :A0
(SCREEN IS ERASED)

          THE DATAWARE MANAGEMENT SYSTEM  (TM)
                                                          M
     1 The DATA DICTIONARY                        B1      M
     2 The PROJECT CHART                          B2      M
     3 The BUILD SYSTEM                           B3      M
     4 Project initialization                     B4      M
     5 SYSTEM HOUSEKEEPING ROUTINES               B5      M

     IS THIS IT? :Y
     DO YOU WISH TO MAKE ADDITIONS,DELETIONS OR CHANGES?(A,D,C) :C
     WHICH SELECTION DO YOU WISH TO CHANGE?
     USE A '*' TO EDIT THE TITLE LINE. :*

(THE MENU.FMT SCREEN IS DISPLAYED HERE)
(THE SCREEN IS ERASED BEFORE AND AFTER THE SCREEN ENTRY)
          THE DATAWARE MANAGEMENT SYSTEM  (TM)
                                                          M
     1 The DATA DICTIONARY                        B1      M
     2 The PROJECT CHART                          B2      M
     3 The BUILD SYSTEM                           B3      M
     4 Project initialization                     B4      M
     5 SYSTEM HOUSEKEEPING ROUTINES               B5      M
     WHICH SELECTION DO YOU WISH TO CHANGE? :

     WHICH MENU DO YOU WISH TO ADD TO, DELETE FROM OR OTHERWISE CHANGE?
     :

(THE SCREEN IS ERASED)

      ********************************************************************************
      *     03/02/82          DATAWARE SYSTEMS  (TM)                                 *
      *------------------------------------------------------------------------------*
      *                       BUILD OR EDIT THE MENUS                                *
      *                                                                              *
      *    1       Build menus from the dictionary          GENMENUS                 *
      *    2       Add or edit menus                        MENU                     *
      *    3       Delete a program call from all menus     DELFUNCT                 *
      *    4       Add a program call to selected menus     ADDFUNCT                 *
      *    5       Print the menus                          PRINTMEN                 *
      *    6       Print the help messages                  PRINTHLP                 *
      *    7       RETURN TO THE BUILD MENU                 B3                       *
      *                                                                              *
      *    >       To do a CP/M command                                              *
      *    .       To do a DBASE II command                                          *
      *    *       To go to a MENU not listed by entering its name                   *
      *    ?       To see help messages for any selection                            *
      *------------------------------------------------------------------------------*
      *  SELECTION         DESCRIPTION                 MENU/FUNCTION NAME           *
      ********************************************************************************
                        ENTER A SELECTION CHARACTER!
     WAITING >
     ENTER the CP/M command.:WS

     *** END RUN     dBASE II     ***
```

Figure 2 Files and Source Listings

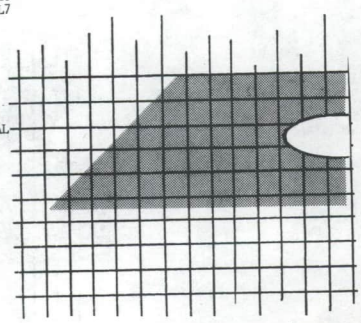
```
STRUCTURE FOR FILE:  MENU.DBF
NUMBER OF RECORDS:   00032
DATE OF LAST UPDATE: 03/03/82
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001       MENU        C    002
002       SELECT      C    001
003       DESCRIPTN   C    040
004       FUNCTION    C    010
005       COMMAND     C    001
006       L0          C    077
007       L1          C    077
008       L2          C    077
009       L3          C    077
010       L4          C    077
011       L5          C    077
012       L6          C    077
013       L7          C    077
** TOTAL **             00671

STRUCTURE FOR FILE:  MENURPT.DBF
NUMBER OF RECORDS:   00003
DATE OF LAST UPDATE: 03/03/82
PRIMARY USE DATABASE
FLD       NAME       TYPE WIDTH   DEC
001       KEY         C    010
002       FILE1       C    010
003       FILE2       C    010
004       KEY1        C    009
005       KEY2        C    009
006       KEY3        C    009
007       INDEXED     L    001
008       FUNCTION    C    010
009       FORMAT1     C    010
010       FORMAT2     C    010
011       LENGTH      N    002
012       MARGIN      N    002
013       TITLE       C    060
014       HEADER      C    132
** TOTAL **             00285
```

```
* MENU.CMD
* 03/09/82
* COPYRIGHT 1982 BY STEVE PATCHEN
* ******** THIS IS THE MENU CONTROL MODULE **********

SET ESCAPE OFF
SET TALK OFF
SET FORMAT TO SCREEN
@ 0,0
@ 1,0 SAY " DATAWARE (TM) MENU SYSTEM COPYRIGHT 1982 BY STEVE PATCHEN        "
@ 2,0
@ 3,0
@ 4,0
@ 5,0 SAY "------------------------------------------------------------------------------"
IF FILE('MENU.MEM')
        RESTORE FROM MENU
        SELECT SECOND
        USE &REPORTFILE
        SELECT PRIMARY
        USE &MENUFILE INDEX &MENUFILE
ELSE
        STORE "MENU" TO MENUFILE
        @ 3,0 SAY 'WHAT IS THE NAME OF THE MENU SYSTEM YOU WISH TO RUN FROM?   GET MENUFILE
        READ
        IF MENUFILE=' '
                STORE 'MENU' TO MENUFILE
                STORE 'MENURPT' TO REPORTFILE
        ELSE
                STORE '&MENUFILE'+'RPT' TO REPORTFILE
        ENDIF
        SELECT SECOND
        USE &REPORTFILE
        SELECT PRIMARY
        USE &MENUFILE INDEX &MENUFILE
        STORE DATE() TO MDATE
        STORE MENU TO CMENU
ENDIF
@ 3,0 SAY "         CORRECT THE DATE IF REQUIRED. " GET MDATE PICTURE "99/99/99"
READ
SET DATE TO &MDATE
SAVE TO MENU
DO WHILE T
        ERASE
        FIND &CMENU
        ? "********************************************************************************"
        ? "*     &MDATE          DATAWARE SYSTEMS  (TM)                                   *"
        ? "*------------------------------------------------------------------------------*"
        DO WHILE .NOT. EOF .AND. CMENU$MENU
                DISPLAY OFF "*    ",SELECT,"     ",DESCRIPTN,FUNCTION,"                  *"
                SKIP
        ENDDO
        ? "*                                                                              *"
        ? "*     >       To do a CP/M command                                             *"
        ? "*     .       To do a DBASE II command                                         *"
        ? "*     *       To go to a MENU not listed by entering its name                  *"
        ? "*     ?       To see help messages for any selection                           *"
        ? "*------------------------------------------------------------------------------*"
        ? "*  SELECTION         DESCRIPTION                 MENU/FUNCTION NAME           *"
        ? "********************************************************************************"
        ? "                       ENTER A SELECTION CHARACTER!"
        WAIT TO ACTION
        DO CASE
        CASE ACTION=">"
                ACCEPT "ENTER the CP/M command." TO LINE
                QUIT TO "&LINE","DBASE MENU"
        CASE ACTION="*"
                ACCEPT "ENTER the two letter MENU name that you want. " TO CMENU
                IF " "$CMENU
                        GOTO 1
                        STORE MENU TO CMENU
                ENDIF
                SAVE TO MENU
        CASE ACTION="."
                ACCEPT "ENTER the DBASE II command. " TO LINE
                IF 0=TEST(LINE)
                        ? "BAD COMMAND LINE!"
                        WAIT
                ELSE
                        &LINE
                ENDIF
                IF FILE("MENU.MEM")
                        RESTORE FROM MENU
                ELSE
                        QUIT TO "DBASE MENU"
                ENDIF
        CASE ACTION="?"
                ACCEPT "ENTER the SELECTION character to get help for. " TO ACTION
                IF .NOT. ACTION=" "
                        IF ACTION="?".OR.ACTION=".".OR.ACTION=">".OR.ACTION="*"
                                LOCATE FOR ACTION$MENU
                        ELSE
                                LOCATE FOR CMENU$MENU.AND.ACTION$SELECT
                        ENDIF
                        IF .NOT. EOF
                        ?
                        ? L0
                        ? L1
                        ? L2
                        ? L3
                        ? L4
                        ? L5
                        ? L6
                        ? L7
                        ELSE
                        ? 'THERE IS NO MESSAGE FOR THIS SELECTION.'
                        ENDIF
                ENDIF
                ? " ENTER any key to continue."
                WAIT
        OTHERWISE
                SKIP -1
                DO WHILE .NOT. SELECT=!(ACTION) .AND. CMENU$MENU
                        SKIP -1
                ENDDO
                IF .NOT. CMENU$MENU
                        ? "NOT A VALID CHOICE. PRESS ANY KEY TO CONTINUE. "
                        WAIT
                ELSE
                        STORE FUNCTION TO MFUNCTION
                        DO CASE
                        CASE COMMAND="C"
                                ERASE
                                ? "USE ^S TO PAUSE DISPLAY"
                                QUIT TO "&MFUNCTION","DBASE MENU"
                        CASE COMMAND="D"
                                ERASE
                                STORE DESCRIPTN TO MFUNCTION
                                QUIT TO "&MFUNCTION","DBASE MENU"
                        CASE COMMAND="F"
                                DO &MFUNCTION
                                RESTORE FROM MENU
                        CASE COMMAND="M"
                                STORE $(MFUNCTION,1,2) TO CMENU
                                SAVE TO MENU
```

```
                CASE COMMAND="E"
                        SELECT SECOND
                        LOCATE FOR "&MFUNCTION"$KEY
                        IF .NOT. EOF
                                STORE FORMAT1 TO PFORMAT
                                STORE FORMAT2 TO SFORMAT
                                STORE FILE1 TO PFILE
                                STORE FILE2 TO SFILE
                                STORE INDEXED TO ISINDEXED
                                STORE KEY1 TO KEYA
                                STORE KEY2 TO KEYB
                                STORE KEY3 TO KEYC
                                STORE TITLE TO ETITLE
                                STORE FUNCTION TO MFUNCTION
                                DO &MFUNCTION
                                RESTORE FROM MENU
                        ELSE
                                ? "&MFUNCTION IS NOT LISTED IN THE SCREEN FORMAT FILE."
                                WAIT
                        ENDIF
                CASE COMMAND="R"
                        SELECT SECOND
                        LOCATE FOR "&MFUNCTION"$KEY
                        IF .NOT. EOF
                                STORE FORMAT1 TO PFORMAT
                                STORE FORMAT2 TO SFORMAT
                                STORE FILE1 TO PFILE
                                STORE FILE2 TO SFILE
                                STORE INDEXED TO ISINDEXED
                                STORE KEY1 TO KEYA
                                STORE KEY2 TO KEYB
                                STORE KEY3 TO KEYC
                                STORE TITLE TO RTITLE
                                STORE HEADER TO RHEADING
                                STORE LENGTH TO FLENGTH
                                STORE MARGIN TO BOT
                                STORE FUNCTION TO MFUNCTION
                                DO &MFUNCTION
                                RESTORE FROM MENU
                        ELSE
                                ? "&MFUNCTION IS NOT LISTED IN THE REPORT FILE."
                                WAIT
                        ENDIF
                ENDCASE
        ENDIF
ENDCASE
IF .NOT. !(ACTION)="M"
  ERASE
  SELECT SECOND
  USE &REPORTFILE
  SELECT PRIMARY
  USE &MENUFILE INDEX &MENUFILE
ENDIF
ENDDO
```

```
* 03/08/82
ERASE
REMARK    MENUEDIT.CMD   COPYRIGHT 1982 BY STEVE PATCHEN
REMARK     THIS ROUTINE ALLOWS EDITING OF MENU FILES
STORE "                                                                  " TO NULLINE
IF SFILE$NULLINE.OR.PFILE$NULLINE
        ? "THE &PFILE PARAMETERS PROVIDED BY THE MENU ARE NOT SUFFICIENT FOR THIS PROGRAM!"
        WAIT
ELSE
  SET FORMAT TO SCREEN
  SELECT SECONDARY
  USE &SFILE
  SELECT PRIMARY
  USE &PFILE INDEX &PFILE
  ? "The &PFILE AND &SFILE are being edited."
  ACCEPT "WHICH MENU DO YOU WISH TO ADD TO, DELETE FROM OR OTHERWISE CHANGE. " TO KEY1VAL
  DO WHILE .NOT. KEY1VAL$NULLINE
     FIND &KEY1VAL
     IF #>0
        STORE # TO POSITION
        ERASE
        ?
        DO WHILE .NOT. EOF .AND.KEY1VAL$MENU
          DISPLAY OFF SELECT,DESCRIPTN,FUNCTION,COMMAND
          SKIP
        ENDDO
        ?
        ACCEPT "IS THIS IT? " TO ANSWER
        IF !(ANSWER)='Y'
                ACCEPT "DO YOU WISH TO MAKE ADDITIONS,DELETIONS OR CHANGES?(A,D,C) " TO ANSWER
                IF !(ANSWER)='C'
                        ? "WHICH SELECTION DO YOU WISH TO CHANGE?"
                        ACCEPT "USE A '*' TO EDIT THE TITLE LINE. " TO KEY2VAL
                        DO WHILE .NOT. KEY2VAL$NULLINE
                                IF KEY2VAL="*"
                                  GOTO POSITION
                                  SET FORMAT TO MENU
                                  READ
                                ELSE
                                  LOCATE FOR KEY1VAL$MENU .AND. KEY2VAL$SELECT
                                  IF .NOT. EOF
                                    SET FORMAT TO MENU
                                    READ
                                    IF COMMAND="R".OR.COMMAND="E"
                                      SELECT SECONDARY
                                      LOCATE FOR KEY=P.FUNCTION
                                      IF EOF
                                        APPEND BLANK
                                        REPLACE KEY WITH P.FUNCTION
                                      ENDIF
                                      SET FORMAT TO MENURPT
                                      READ
                                SELECT PRIMARY
                              ENDIF
                            ENDIF
                        ENDIF
                        ERASE
                        LIST OFF FOR KEY1VAL$MENU SELECT,DESCRIPTN,FUNCTION,COMMAND
                        ACCEPT "WHICH SELECTION DO YOU WISH TO CHANGE? USE '*' TO DO THE TITLE LINE." TO KEY2VAL
                ENDDO
        ELSE
        IF !(ANSWER)='D'
                ACCEPT "WHICH SELECTION DO YOU WISH TO DELETE? USE '*' TO DELETE IT ALL." TO KEY2VAL
                IF KEY2VAL='*'
                        DELETE FOR KEY1VAL$MENU
                        PACK
                        ? KEY1VAL+' HAS BEEN DELETED.'
                ELSE
                  LOCATE FOR KEY1VAL$MENU.AND.KEY2VAL$SELECT
                  IF EOF
                        ? 'I CANNOT FIND THIS ONE.'
                        WAIT
                ELSE
                        ? DESCRIPTION,FUNCTION,COMMAND
                        ACCEPT 'IS THIS IT?' TO ANSWER
                        IF !(ANSWER)='Y'
                                DELETE
                                PACK
                                IF COMMAND='R'.OR.COMMAND='E'
                                  SELECT SECOND
                                  LOCATE FOR KEY=P.FUNCTION
                                  IF .NOT. EOF
                                        DELETE
                                        PACK
                                  ENDIF
                                  SELECT PRIMARY
                                ENDIF
                        ENDIF
                  ENDIF
                ENDIF
        ELSE
        IF !(ANSWER)='A'
                ACCEPT "WHAT SELECTION DO YOU WISH TO ADD? " TO KEY2VAL
                ACCEPT "WHICH EXISTING SELECTION WILL THIS FOLLOW? " TO ANSWER
                LOCATE FOR KEY1VAL$MENU.AND.ANSWER$SELECT
                IF EOF
                        ? "I DO NOT FIND THIS ONE."
                        WAITING
                ELSE
                        INSERT BLANK
                        REPLACE MENU WITH KEY1VAL,SELECT WITH KEY2VAL
                        SET FORMAT TO MENU
                        READ
                        IF COMMAND='R'.OR.COMMAND='E'
                          SELECT SECOND
                          APPEND BLANK
                          REPLACE KEY WITH P.FUNCTION
                          SET FORMAT TO MENURPT
                          READ
                          SELECT PRIMARY
                        ENDIF
                ENDIF
        ELSE
                ? " CHOICE NOT IMPLEMENTED YET."
                WAIT
        ENDIF
        ENDIF
        ENDIF
    ENDIF
ELSE
    ACCEPT "I CANNOT FIND IT. DO YOU WISH TO CREATE IT? " TO ANSWER
    IF !(ANSWER)='Y'
        ACCEPT "HOW MANY SELECTIONS WILL BE IN THE MENU? " TO ANSWER
        STORE VAL(ANSWER) TO COUNT
        STORE 1 TO SELECTNUM
        ACCEPT " ENTER THE MENU TITLE LINE. " TO ANSWER
        APPEND BLANK
        REPLACE MENU WITH KEY1VAL,DESCRIPTN WITH ANSWER
        APPEND BLANK
        REPLACE MENU WITH KEY1VAL
        DO WHILE COUNT>0
                APPEND BLANK
                REPLACE MENU WITH KEY1VAL,SELECT WITH STR(SELECTNUM,1)
                SET FORMAT TO MENU
                READ
                IF COMMAND='R'.OR.COMMAND='E'
                  SELECT SECOND
                  APPEND BLANK
                  REPLACE KEY WITH P.FUNCTION
                  SET FORMAT TO MENURPT
                  READ
                  SELECT PRIMARY
                ENDIF
                STORE COUNT-1 TO COUNT
                STORE SELECTNUM+1 TO SELECTNUM
        ENDDO
        LIST OFF FOR KEY1VAL$MENU SELECT,DESCRIPTN,FUNCTION,COMMAND
    ENDIF
  ENDIF
  ERASE
  ?
  ? "WHICH MENU DO YOU WISH TO ADD TO, DELETE FROM OR OTHERWISE CHANGE? "
  ACCEPT TO KEY1VAL
ENDDO
RELEASE ANSWER,KEY1VAL,KEY2VAL,NULLINE
ENDIF
RETURN
```

```
* 03/03/82
* MENU.FMT
* COPYRIGHT 1982 BY STEVE PATCHEN
@ 2,0  SAY '********** MENU EDITOR *************'
@ 3,0
@ 4,0  SAY 'menu name        ' +KEY1VAL
@ 5,0  SAY '   menu selection     ' GET SELECT
@ 6,0
@ 7,0  SAY 'selection description  ' GET DESCRIPTN
@ 8,0  SAY 'selection function     ' GET FUNCTION
@ 9,0  SAY '  COMMAND CHARACTER     ' GET COMMAND
@ 10,0
@ 11,0 SAY 'THE HELP MESSAGE FOR THIS SELECTION IS:'
@ 12,0
@ 13,0 GET L0
@ 14,0 GET L1
@ 15,0 GET L2
@ 16,0 GET L3
@ 17,0 GET L4
@ 18,0 GET L5
@ 19,0 GET L6
@ 20,0 GET L7
```



(continued next page)

```
* 03/03/82
* MENURPT.FMT
* COPYRIGHT 1982 BY STEVE PATCHEN
@ 2,0  SAY '********** REPORT AND SCREEN PARAMETERS *************'
@ 3,0
@ 4,0  SAY 'menu name        ' +KEY1VAL
@ 5,0  SAY '    menu selection     ' +KEY2VAL
@ 6,0
@ 7,0  SAY 'selection description  ' +DESCRIPTN
@ 8,0  SAY 'selection function     ' +FUNCTION
@ 9,0  SAY '  COMMAND CHARACTER    ' +COMMAND
@ 10,0
@ 11,0 SAY ' files used            ' GET FILE1
@ 11,50 GET FILE2
@ 12,0 SAY 'are the files indexed?' GET INDEXED
@ 13,0 SAY ' keys used       ' GET KEY1
@ 13,40 GET KEY2
@ 13,55 GET KEY3
@ 14,0 SAY 'screen or report form files if required'
@ 15,0 GET FORMAT1
@ 15,40 GET FORMAT2
@ 16,0 SAY 'report or entry function name'    GET FUNCTION
@ 17,0 SAY 'for reports, enter the form length and margin allowance.
@ 18,0  GET LENGTH
@ 18,30 GET MARGIN
@ 19,0 SAY 'if there is a title or heading enter it.
@ 20,0 GET TITLE
@ 21,0 GET HEADER
```

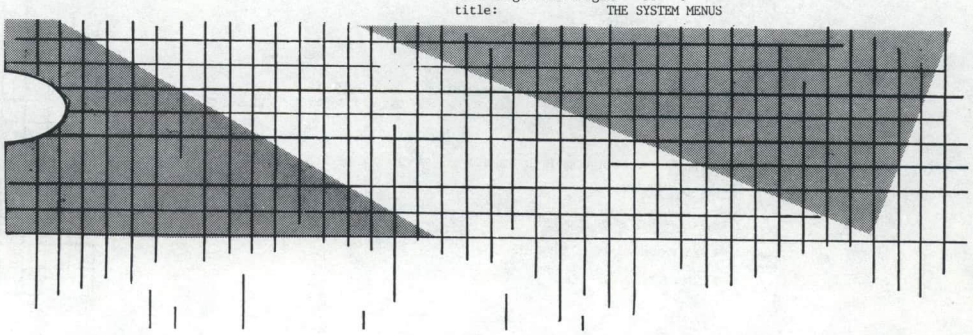
```
* 03/03/82
* REPORT1.CMD prints a group title and then calls PFORMAT
* uses: RTITLE,RHEADING,PAGE,LINES,FLENGTH,PFILE,SFILE,KEYA,KEYB,VALUE
*       PFORMAT,SFORMAT,NULLINE,KEYAVAL,KEYBVAL,BOT
* calls: TOPOFPAG

STORE "                                                    " TO NULLINE
IF .NOT. SFILE$NULLINE
       SELECT SECOND
       USE &SFILE
ENDIF
SELECT PRIMARY
USE &PFILE
STORE 1 TO PAGE
STORE TRIM(RHEADING) TO RHEADING
SET FORMAT TO PRINT
SET PRINT ON
DO WHILE .NOT. EOF
       DO TOPOFPAG
       DO WHILE .NOT. EOF .AND.(LINES-BOT-3)>0
              STORE &KEYA TO KEYAVAL
              ?
              ? KEYA,KEYAVAL
              STORE LINES -2 TO LINES
              DO WHILE .NOT. EOF .AND. &KEYA=KEYAVAL
                     STORE &KEYB TO KEYBVAL
                     DO &PFORMAT
                     SKIP
                     IF (LINES-BOT)<1
                            DO TOPOFPAG                    `P
                     ENDIF
              ENDDO
       ENDDO
ENDDO
RELEASE PFILE,SFILE,PFORMAT,SFORMAT,KEYA,KEYB,KEYC,RTITLE,RHEADING,FLENGTH
RELEASE PAGE,LINES,NULLINE,KEYAVAL,KEYBVAL,KEYCVAL,BOT
SET FORMAT TO SCREEN
EJECT
EJECT
SET PRINT OFF
RETURN
```

```
* 03/03/82
* MENURPT1.CMD prints menu entries
* uses: KEYB,KEYBVAL,LINES,BOT,LCNT,LINE,KEYA,KEYAVAL,NULLINE
* calls: TOPOFPAG

?
? "            ",KEYBVAL,"   ",DESCRIPTN,FUNCTION,COMMAND
STORE LINES -2 TO LINES
IF COMMAND="R".OR.COMMAND="E"
       IF (LINES-BOT-9)<1
              DO TOPOFPAG
       ENDIF
       STORE LINES-9 TO LINES
       SELECT SECOND
       LOCATE FOR KEY=P.FUNCTION
       IF .NOT. EOF
        ? "                          report or screen key: ",KEY
        ? "                                         files: ",FILE1,FILE2
        ? "                                     file keys: ",KEY1,KEY2,KEY3
        ? "                            files are indexed: ",INDEXED
        ? "                                function file: ",FUNCTION
        ? "                                 format files: ",FORMAT1,FORMAT2
        ? "                           form length and margin: ",LENGTH, MARGIN
        ? "                          title: ",TITLE
        ? "                        heading: ",HEADER
       ELSE
        ?
        ?
        ? "             format record not found!"
        ?
        ?
        ?
        ?
        ?
       ENDIF
       SELECT PRIMARY
ENDIF
STORE 0 TO LCNT
STORE "LO" TO LINE
DO WHILE LCNT < 8 .AND. .NOT. &LINE$NULLINE
       IF (LINES-BOT)<1
              DO TOPOFPAG
       ENDIF
    .  ? "              ",&LINE
       STORE LCNT+1 TO LCNT
       STORE "L"+STR(LCNT,1) TO LINE
       STORE LINES -1 TO LINES
ENDDO
RELEASE LCNT,LINE
RETURN
```

```
* 03/08/82
* TOPOFPAG.CMD   does an eject and a new top margin heading
* uses : RTITLE, RHEADING, PAGE, LINES, FLENGTH
EJECT
STORE FLENGTH TO LINES
? DATE()+"                                                    page "+STR(PAGE,2)
? "          "+RTITLE
? RHEADING
STORE "---(this line is suppose to have 132 '-'s in it)---" TO RLINE
? $(RLINE,1,LEN(RHEADING))
?
STORE LINES-5 TO LINES
STORE PAGE+1 TO PAGE
RELEASE RLINE
RETURN
```

Figure 3 A Menu Report

```
                              THE SYSTEM MENUS
MENU NAME, SELECTION,          DESCRIPTION,                    FUNCTION, COMMAND
-------------------------------------------------------------------------------


MENU      A0

                     THE DATAWARE MANAGEMENT SYSTEM  (TM)

                                                                          M

          1     The DATA DICTIONARY                        B1           M
           THE DATA DICTIONARY MENU HAS ROUTINES FOR ENTERING AND REPORTING DICTIONARY
          ENTRIES.

          2     The PROJECT CHART                          B2           M
            THE PROJECT CHART MENU CONTAINS ROUTINES TO CREATE AND EDIT THE PROGRESS
          CHART AND THE WORK ASSIGNMENTS. IT ALSO HAS ROUTINES TO PRINT PROGRESS
          REPORTS AND SYSTEM FORMS.

          3     The BUILD SYSTEM                           B3           M
            THE BUILD SYSTEM MENU HAS ROUTINES TO BUILD AND MAINTAIN DATA PROCESSING
          APPLICATIONS. IT HAS ROUTINES TO BUILD THE SYSTEM MANUALLY OR AUTOMATICALLY
          FROM A LOGICAL STRUCTURE IN THE DATA DICTIONARY.

          4     Project initialization                     B4           M
            THIS MENU CONTAINS ROUTINES WHICH ARE REQUIRED WHEN STARTING A NEW PROJECT.

          5     SYSTEM HOUSEKEEPING ROUTINES               B5           M
            THIS MENU CONTAINS ROUTINES TO BACKUP SYSTEM FILES AND TO DO OTHER GENERAL
          MAINTENANCE ON THE SYSTEM.

MENU      B3

                          THE BUILD SYSTEM                                M

          1     Build or Edit the Menus                    M0           M


                              THE SYSTEM MENUS
MENU NAME, SELECTION,          DESCRIPTION,                    FUNCTION, COMMAND
-------------------------------------------------------------------------------

          2     Build data files                           N0           M

          3     Build Reports                              R0           M

          4     Build entry screens                        S0           M

          5     Build Program modules                      P0           M

          6     RETURN TO THE MAINMENU                     A0           M

MENU      M0
                        BUILD OR EDIT THE MENUS


          1     Build menus from the dictionary            GENMENUS     F

          2     Add or edit menus                          MENU         E
                 report or screen key:  MENU
                                files:  MENU       MENURPT
                            file keys:  MENU       SELECT
                       files are indexed:  .T.
                           function file:  MENUEDIT
                            format files:  MENU       MENURPT
                  form length and margin:  24   2
                  title:              THE SYSTEM MENUS
                 heading:

          3     Delete a program call from all menus       DELFUNCT     F

          4     Add a program call to selected menus       ADDFUNCT     F


                              THE SYSTEM MENUS
MENU NAME, SELECTION,          DESCRIPTION,                    FUNCTION, COMMAND
-------------------------------------------------------------------------------


          5     Print the menus                            REPORT1      R
                 report or screen key:  REPORT1
                                files:  MENU       MENURPT
                            file keys:  MENU       SELECT
                       files are indexed:  .T.
                           function file:  REPORT1
                            format files:  MENURPT1
                  form length and margin:  51   8
                  title:              THE SYSTEM MENUS
```



```
   heading:  MENU NAME, SELECTION,          DESCRIPTION,                    FUNCTION, COMMAND
THIS ROUTINE USES A REPORT FUNCTION TO PRINT A COPY OF THE SYSTEM MENUS.

6     Print the help messages                    PRINTHLP   F

7     RETURN TO THE BUILD MENU                   B3         M

8     Print the menus to a file                  REPORTFI   R
      report or screen key:  REPORTFI
                     files:  MENU       MENURPT
                 file keys:  MENU       SELECT
            files are indexed:  .T.
              function file:  REPORTFI
                format files:  MENURPT1
      form length and margin:  51  12
                     title:              THE SYSTEM MENUS
   heading:  MENU NAME, SELECTION,          DESCRIPTION,                    FUNCTION, COMMAND
THIS ROUTINE PRINTS TO A FILE INSTEAD OF THE PRINTER



MENU     *

              THIS IS THE DIRECT ROUTING MENU COMMAND
         THE * SELECTION IS USED TO JUMP DIRECTLY TO A MENU BY ENTERING ITS NAME.
         Respond to the prompt with the two character name of the menu. This name is

                        THE SYSTEM MENUS
MENU NAME, SELECTION,          DESCRIPTION,                    FUNCTION, COMMAND
-------------------------------------------------------------------------------

         displayed to the right of the selection descriptions. Those selections with
         longer names are functions. They can be called with the '.' selection.

MENU     ?

              THIS IS THE COMMAND TO DISPLAY HELP MSGS
          THE ? SELECTION IS USED TO VIEW HELP MESSAGES FOR ANY SELECTION IN THE MENU.
         Respond to the prompt with the character for the selection you wish to view.

MENU     .

              THIS IS THE DBASE II DIRECT COMMAND
         THE '.' SELECTION CAN BE USED TO ENTER ANY DBASE II COMMAND LINE.
         Respond to the prompt with the command line you wish executed. The syntax of
         the line will be tested before executing it. This selection can be used to
         execute functions directly by using the 'DO' command.

MENU     >

              THIS IS THE CP/M DIRECT COMMAND ENTRY
         THE '>' SELECTION ALLOWS EXECUTION OF CP/M COMMANDS. THE MENU IS RESTORED
         AFTER THE CP/M COMMAND IS COMPLETED. Respond to the prompt with the command
         line you wish executed.
```


(continued next page)

8080 Assembler Programming Tutorial: Pitfalls of Programming

Ward Christensen

When starting to program in 8080 assembly language, you might be tempted to use some instruction sequences which would not have the expected effect. I learned these the hard way - by writing programs that just didn't work.

Also, there are times when debugging a program under DDT or SID, can cause unforeseen problems.

I will attempt to address a few of these areas to save you the time of stumbling into them yourselves. You will also find the many *Tips and Techniques* columns in *Lifelines* useful. This tutorial appropriately deals with the simpler initial pitfalls that the novice is likely to encounter.

Setting The Condition Code

The 8080 condition code bits in the PSW, such as zero, sign, parity, are set only by certain instruction classes: arithmetic (ADD, SUB, DAA, etc.), logical (AND, ANA, ORA etc.), and rotating (RAL, RRC, etc). The pitfall: data *movement* instructions do *not* set the condition bits. For example,

```
;
;This does not work:
;
     INX  H      ;point to next
     MOV  A,M    ;get char
     JZ   FINAL  ;was it zero?
```

It doesn't work, because the MOV A,M didn't set the condition code bits. I have even seen this coded in a book which purported to teach how to program. The author apparently knew 6800 or 6502, and just made wrong assumptions about the 8080.

How *do* you make such an instruction sequence work? Just code an ORA A instruction, which ORs the accumulator with itself (thus not changing anything) and sets the condition code bits:

```
INX  H      ;point to next
MOV  A,M    ;get char
ORA  A      ;was it zero?
JZ   FINAL  ;yes, branch.
```

Any of several instructions may be used to make the test. I prefer ORA A, but ANA A would serve as well. Also, a compare instruction could be used - CPI 0 - but "hackers" like to keep in mind that CPI 0 takes two bytes, and ORA A only one.

CONCLUSION: Watch which instructions set the condition code bits. In general, data movement instructions do not.

Comparing Values

When comparing two values, the condition code bits are set based upon the results. The same is true for subtraction. Suppose you want to know whether one value is less than another. You might be tempted to use the *sign* PSW bit, which is tested with JM, jump if minus. However, when the values are greater than 127, the sign bit just doesn't properly reflect the outcome.

Here is a condensed listing of what SID produces when the values 23H and E8H are compared with 10H, 23H, 7EH, 0C0H, 0E8H, and 0FFH. The left three columns show the state of the (C)arry, (Z)ero, and (M)inus flags in the PSW. Note that the flags are shown on the line *after* the line showing the instruction. For example, I loaded a 23 into the accumulator at address 100, and at 104 compared it to a 23. The *next* line shows the Z flag on, i.e. Zero, indicating an equal compare.

```
--- 0100 MVI  A,23
--- 0102 CPI  10
--- 0104 CPI  23
-Z- 0106 CPI  7E
C-M 0108 CPI  C0
C-- 010A CPI  E8
C-- 010C CPI  F9
C-- 010E MVI  A,E8
C-- 0110 CPI  10
--M 0112 CPI  23
--M 0114 CPI  7E
--- 0116 CPI  C0
--- 0118 CPI  E8
-Z- 011A CPI  F9
C-M 011B RST  7
```

All the values shown are in hexadecimal, so I won't show them as 23H, but just 23.

Let's look at the individual instructions and their results. At 100, I loaded 23 (hex) into A. In comparing it to 10 at 102, no indicators were set, as shown in the line at address 104. Then, you see the compare to 23 set (Z)ero, as would be expected.

The compare to 7E resulted in (C)arry and (M)inus being set. Since the CPI instruction is like a subtract instruction that sets the indicators but doesn't change the accumulator, you would expect M to be set on: 23-7E produces a negative result.

You might also expect the compare to C0 to set (M)inus, but there's a catch. The (M)inus indicator is set as if the values were signed. Hex C0, if treated as signed, is -40. Now, if you consider the CPI to set the status bits like a subtract, subtracting C0 becomes a subtract of -40. To subtract a negative number, you complement it and add. Thus it becomes 23 + 40. This is 63, a positive number, so the (M)inus flag is *not* on.

How can you avoid this pitfall when you don't know the values you will be working with? Unless you are *explicitly* working with *signed* numbers, just use the (C)arry flag. As I mentioned in section 8 of this tutorial, I have a little memory aid for the state of (C)arry after a compare: "C.A.L.", which stands for "Carry if Accumulator is Less". This works for all values, as you see in the example. For instance, in MVI A,E8 and its following compares, only the compare with F9 sets carry, since that was the only value for which the accumulator (E8) was less.

CONCLUSION: Use (C)arry to indicate comparisons in which the accumulator is less.

Counting in Registers

The most common way to repeat a sequence of instructions a given number of times is to set a count in a register, and count it down to zero. For example:

```
      MVI   C,8   ;REPEAT 8 TIMES
LOOP  MOV   A,M   ;GET CHAR
      STAX  D     ;STORE IT
      .
      .
      .
      DCR   C     ;MORE?
      JNZ   LOOP  ;YES, LOOP
```

This technique works for values from 1 to 255, and, if a value of 0 is used, the loop is repeated 256 times. This is because it will first be decremented to 255, so the entire process will be repeated 256 times.

When you want to count to values greater than 256, you will need to use two registers. One way is to decrement a register pair:

```
DCX  B
```

However, the condition code bits are *not* set by this operation, so you must explicitly test for zero by:

```
MOV  A,B  ;GET B
ORA  C    ;COMBINE WITH C
```

before doing the:

```
JNZ  LOOP ;LOOP UNTIL BC=0
```

It would be tempting to use DCR instructions, and they can be made to work, but perhaps not in an obvious manner:

```
;
;This does not work
;

  DCR  C    ;DCR LOW
  JNZ  LOOP ;NOT ZERO?
  DCR  B    ;DCR HIGH
  JNZ  LOOP ;NOT ZERO?
```

The reason this doesn't *not* work, is that when C is not 00, B is actually one too small. Only when C *is* 00 does it work. For example, if you wish to repeat something 200 hex times, you can:

```
      LXI   B,200H
LOOP  .
      .
      DCR   C
      JNZ   LOOP
      DCR   B
      JNZ   LOOP
```

It works because the first DCR C loop will be repeated 100H times, then DCR B will go from 2 to 1, and the DCR C will be executed another 100H times, a total of 200H. Suppose however that the loop is to be 102H times:

```
      LXI   B,102H
LOOP  .
      .
      DCR   C
      JNZ   LOOP
      DCR   B
      JNZ   LOOP
```

The first DCR C loop will be repeated two times, then DCR B will go from 1 to 0, and the loop will not be executed further. Thus the loop was executed only two times.

There are times when you are trying to make a program work as rapidly as possible. In that case, the technique of

```
DCX  B
MOV  A,B  ;GET B
ORA  C    ;COMBINE WITH C
JNZ  LOOP ;LOOP UNTIL BC=0
```

is a bit slower than the

```
DCR  C
JNZ  LOOP
DCR  B
JNZ  LOOP
```

technique. So, what is needed, is a "front end" for the DCR technique, which will make sure it works properly for all cases.

Let's take a character move subroutine as an example, and add the front end to make it *fast*. First, the traditional move routine:

```
;Move subroutine.
;Enter with (HL)=from, (DE)=to,
;     and (BC)=count.
;
MOVE  MOV   A,M   ;GET CHAR
      STAX  D     ;SAVE IT
      INX   H     ;BUMP POINTER
      INX   D     ;BUMP POINTER
      DCX   B     ;DEC. COUNT
      MOV   A,B   ;IS COUNT
      ORA   C     ;    =0?
      JNZ   MOVE
      RET
```

The inner loop takes 50 machine cycles, which on a 4MHz system means 12.5 microseconds per byte.

```
;
;Move subroutine.
;Optimized for speed
;Enter with (HL)=from, (DE)=to,
;     and (BC)=count.
;
;First, increment B when C is
;not zero, so DCR looping may
;be done.
;
MOVE  MOV   A,C   ;CHECK LOW COUNT
      ORA   A     ;ZERO?
      JZ    MV2
      INR   B     ;FUDGE B
;
MV2   MOV   A,M   ;GET CHAR
      STAX  D     ;SAVE IT
      INX   H     ;BUMP POINTER
      INX   D     ;BUMP POINTER
      DCR   C     ;DECREMENT
      JNZ   MOVE  ;DONE?
      DCR   B     ;DECREMENT
      JNZ   MOVE  ;DONE?
      RET
```

Admittedly there is a bit of set-up time, but the inner loop time is now reduced to 40 clock cycles, or 10 microseconds.

NOTE: On a Z-80, the LDIR instruction accomplishes this exact task in the processor hardware, in one instruction. I.E. instead of:

```
CALL MOVE
```

you could simply use::

```
LDIR
```

However, this will restrict your code to running on a Z-80 only. It is faster – 21 cycles, or 5.25 microseconds. If you are running the standard CP/M ASM or MAC, LDIR is not a recognized operation code, so you will have to code its value explicitly:

```
DB    0EDH,0B0H ;Z-80 LDIR
```

CONCLUSION: Just be aware of the pitfalls of 16-bit counting.

Complementing

The CMA instruction complements each individual bit of a register. To take the negative of a register requires a CMA followed by an INR. INR adds one to the register; thus with CMA it forms the proper negative of the original value.

(continued next page)

You can similarly make the negative of a 16-bit value, but this time you increment the *entire* value after CMAing it:

```
MOV   A,B   ;GET HIGH
CMA         ;COMPLEMENT
MOV   B,A   ;PUT IT BACK
MOV   A,C   ;GET LOW
CMA         ;COMPLEMENT
MOV   C,A   ;PUT IT BACK
INX   B     ;ADD 1 TO BC
```

CONCLUSION: Remember CMA only flips the bits, and you must increment the value if you want the *negative* of the original.

Setting the Stack Under DDT Or SID

When you write programs, they will typically either return to CP/M or warm boot when finished. To return to CP/M, you must save the stack which is passed to you by CP/M's console command processor (CCP), and then restore it before returning.

When testing a program under DDT or SID, there is *no* stack set for you. Thus, if your program is to return to CCP, you must set the stack first. Here is a suitable technique if you have some high memory available:

```
F800    LXI   SP,F900
F803    CALL  100
F806    RST   7
```

With this technique, the stack is set at F900, and the CALL 100 causes your program to be executed, in a manner similar to being called from CCP. When it returns, it goes to F806, where the RST 7 returns to DDT or SID.

Obvious Pitfalls

BALANCE STACK USAGE. For every PUSH, there must be a POP, and for every CALL, a RETurn.

KEEP PUSHES AND POPS IN ORDER. It is very tempting to do:

```
;
;this does not work
;
;SAVE REGS
;
      PUSH B
      PUSH D
      PUSH H
      .
      .
      .
      POP  B
      POP  D
      POP  H
```

It doesn't work because after the three PUSHes, HL is the topmost entry on the stack, and the POP B POPs what was in HL, into BC. The POPS must be:

```
POP  H
POP  D
POP  B
```

START YOUR COMMAND FILES AT 100H. Command files (.COM) under CP/M run at 100H, not at zero. If you do not place an ORG statement at the start of your file, DDT, or LOAD will screw up. Place:

```
ORG  100H
```

before any other instructions (except EQU or comments) in your program.

LOAD THE STACK POINTER if you are going to use it in your program. (Very few programs *don't* use the stack). For example, near the start of your program:

```
LXI   SP,STAK
```

then reserve the area:

```
      DS    100  ;100 bytes
STAK  EQU   $
```

NOTE the stack works its way *down*, so the label *follows* the stack allocation.

DOCUMENT YOUR PROGRAMS. It's nice to go back some time later, and not have to *guess* what your program did, or when it was written. A generally good program introduction would include:

```
Program name:
Author:
Date written:
Usage:
Execution:
Dependencies:
  Hardware:
  Software:
Modifications:
```

DESK CHECK YOUR PROGRAMS. I don't recommend doing this *every* time, but very few programmers write programs that run the first time, or that have no latent bugs. If I have a sequence of instructions that I'm are not certain of, I like to "play computer". To do so, I take a piece of paper, and make columns for each of the registers, and the stack:

A	B C	D E	H L	STACK

By grouping B and C, D and E, and H and L near each other, I can either write in the single byte values that each contains, or I can write a 16-bit value for the register pair.

For example, if the first instruction in my program was:

```
LXI H,0
```

my paper would look like:

A	B C	D E	H L	STACK
			0000	

If I then said:

```
MVI D,7
```

I'd have:

A	B C	D E	H L	STACK
		7		0000

After:

```
PUSH H
```

I'd have:

A	B C	D E	H L	STACK
		7	0000	0000

etc. Note that for the registers, I show their actual *content*, but for the stack, I show not the contents of the stack *register*, but what has been pushed *onto* the stack. When I then POP something off of the stack, I cross it off. Thus the entries under STACK represent the items currently in it, and the order they are in is correct. When a routine ends, typically the stack should again be empty.

This technique has helped me uncover many potential bugs: unbalanced stack usage, registers getting clobbered, etc.

Other Pitfalls

Have you discovered any other novice-oriented pitfalls? Send them in to me care of *Lifelines*, and I'll collect and print them in a future tutorial column.

A Detailed Description of PLAN80, Part 1

Raymond J. Sonoff

PLAN80 is a planning and analysis tool from Business Planning Systems, Inc. of Dover, Delaware, distributed by Lifeboat Associates. Preliminary descriptions of PLAN80 have appeared in *Lifelines* (on page 35 of the 'New Products' section in the September 1981 issue and under the 'New Versions' column on page 44 of the December 1981 issue), but no detailed description has been published here. The purpose of this article is to provide an introductory user's viewpoint of PLAN80. Topics to be addressed include the following. What is needed to set up PLAN80 for a given system? What are some key features associated with having PLAN80 as an operational software tool? How useful are the examples provided in the manual? And, is PLAN80 really worth considering for my own particular areas of application of computer-based modeling?

A subsequent article will present specific examples of modeling using PLAN80 control statements and will illustrate how this product can be used to provide sensitivity analyses, and produce finished reports, hopefully while improving an individual's understanding, productivity, and overall effectiveness.

What Is PLAN80?

PLAN80 is a modeling system that helps you plan, forecast, project, estimate, analyze, control, and understand numbers representing sales, profits, costs, taxes, cash, marketing plans, cost center expense, R and D projects, market share, growth rates, return on investment, capital projects, real estate deals, and discounted cash flow.

In short, PLAN80 is a computer-based planning tool that the user can configure for his particular needs and desires, thereby eliminating drudgery previously associated with calculating, recalculating, checking and ultimately producing useful reports.

Background

PLAN80 is a PLANning tool that works with either 8080- or Z80-based computer systems. Aside from requiring an Editor (of your own choice) to create and modify text files, PLAN80 is a standalone software package. Minimum system requirements include the following: an 8080- or Z80-based computer system, a CP/M operating system, 56K of RAM, a console with clear screen and cursor addressing functions, an Editor, a diskette with CP/M on it, one disk drive having at least 100K bytes of storage (or two drives, if you wish to copy files), and the operating manual from Business Planning Systems, Inc.

Since every tabular report has certain common elements, PLAN80 uses a language that instructs your computer to process labeled information. This will involve TITLES, the framework of the report in terms of ROWS and COLUMNS, the starting DATA values, and the RULES used to compute totals and other values. Supplementary information that you can provide PLAN80 includes column width, line spacing, the number of decimal positions, etc., so that the report becomes visually effective.

Highlights of PLAN80

Manual. The 138-page manual is supplied in a 3-ring binder and is organized as shown below into eight chapters and five appendices. An index is also provided.

Chapter 1: Introduction
Chapter 2: PLAN80 Reference Overview
Chapter 3: Defining the Framework
Chapter 4: Starting Values
Chapter 5: Calculations
Chapter 6: Output Statements
Chapter 7: Communicating between Applications
Chapter 8: Other Statements
Appendix A: PLAN80 Examples
Appendix B: PLAN80 Installation
Appendix C: Operating PLAN80
Appendix D: PLAN80 Error Messages
Appendix E: Size Considerations

Control statements. PLAN80 models are comprised of control statements of any of the following:

:TITLES	Heading at top of page
:COLUMNS	Columnar structure
:ROWS	Row structure
:DATA	Actual or assumed values
:RULES	Calculations
:FOR	Scope of other statements
:PUT	Writes values to disk
:GET	Reads values from disk
:INITIALIZE	Sets all values to zero
:INTERACTIVE	Where to begin recalculating
:PRINT	Print reports automatically
$:OPTIONS	Control report appearance
$:DISPLAY	View results on CRT screen Graph results on screen Add or change values interactively Specify print options interactively Print reports on screen, printer, or disk
:INCLUDE	Include statements from a separate file
:REPEAT	Repeatedly include a separate file
:MODELSIZE	Determine maximum number of rows and columns

Note: Only the COLUMNS and ROWS statements are mandatory. All

(continued next page)

other statements are optional.

Functions. Twenty-four mathematical functions are provided by PLAN80. Each function is recognized by PLAN80 when preceded by an "@" symbol. Among the functions available are these:

@SUM(jan..jun)	sum of groups of rows or columns
@AVG(row6..row12)	averages
@CUM(income)	cumulative sum within a row or column
@MAX((0,profit)	largest of a list of values
@MIN(alt1,alt2,alt3)	smallest of a list of values
@INT(col7)	integer portion of a number
@FRAC(col7)	fractional portion of a number
@COS(angle3)	trigonometric functions of angles expressed in radians
@LOOKUP(income,bracket1..bracket5,rate1)	table lookup
@IRR(cashflow)	internal rate of return
@SL(amount,life)	straight line depreciation
@SOD(amount,life)	sum-of-digits depreciation
@DB(amt,life,1.25)	declining balance depreciation
@DBXSL(amt,life,2.)	declining balance/straight line

The symbols within the parentheses are simply representative "names" that illustrate the flexibility of PLAN80 coding.

Error Codes. When PLAN80 encounters a statement whose syntax is incorrect for some reason, you will observe an error message on the CRT screen. Most messages indicate that PLAN80 was expecting an equal sign, a closing parenthesis, a valid name, etc., but found something else. There are more than thirty error codes provided to assist you in determining the nature of any syntax problems which occur in programs you have created. These error codes are explained both in the manual and on the Reference Card.

PLAN80 will also tell you where the error was encountered in your program through use of a "↑" symbol immediately below or to the right of the source of error. It is possible, however, that an entry in prior lines may be the cause of the problem even though it was syntactically correct - but not what you intended - and resulted in the noted error. Whether you wish to correct the error or to ignore it is left to you. Correcting all syntax problems associated with input statements and beginning again is certainly the best approach when actual printout of results is desired.

Hard copy. Regardless of your application, you will probably want the results presented as a tabular report. PLAN80 automates this task for you. Thus, long range planning, capital project evaluation, balance sheet projections, project budgeting and control, profit and loss, cash flow, and numerous other computations can be readily accomplished (including recalculation for other sets of data, if so desired), and each application of PLAN80 can be summarized as a tabularized printed report using the PRINT statement.

Getting Started

After an initial overview of the manual, especially of Chapters 1 and 2, I decided to configure my system following the steps cited in Appendix B. For, once this was accomplished, I could try out Appendix C: Operating PLAN80, and also be able to more readily appreciate the seven examples provided in Appendix A.

Within an hour after receiving my software package I had PLAN80 up and running. The normal procedures of blank diskette formatting, SYSGENning of CP/M, and copying of PIP.COM were followed by PIPping PLAN80.* files onto this diskette from the diskette received with the PLAN80 manual. This Category I file is comprised of PLAN80.COM and PLAN80.OVR files. Next, Category II files were PIPped using the PIP A:=B:*.TXT command. These seven files are examples of PLAN80 applications that will be described later.
Lastly, to configure the console terminal portion of PLAN80, I selected the ZENITH.TRM file (since I have a Zenith WH19/Heath H19 terminal as my console input) from the TERMINAL.LST menu of seven terminal types. After performing a PIP CONTROL.TRM=B:ZENITH.TRM command, I was immediately able to try out PLAN80's examples described in Appendix A of the manual. Everything worked perfectly!

Special note: although I did not have to use it, an excellent INSTALL.COM program is provided with the original diskette to assist you in configuring non-standard terminals (defined here as any terminal other than one included in the TERMINAL.LST menu). Nearly six pages of description are devoted to explaining how to implement and check out each step of the INSTALL program, and a full recovery mode of testing is built into each test. There are five tests: cursor addressing, clear screen and home, clear to end of line, clear to end of screen, and highlighting. When you can answer the question, "Was test successful?" for the first test, you automatically advance to the next test, and so on. I tried out the INSTALL program using the Read Current File command (resulting in reading of the CONTROL.TRM file that is identical to the ZENITH.TRM file). I was impressed by the interactive "console self-test" program that clearly functioned properly as proof of proper configuration. Much thought went into this INSTALL program, and it will certainly prove a boon to non-standard terminal owners who wish to get PLAN80 up and running quickly.

Examples. In Appendix A of the PLAN80 manual are the following:

2. Projected Sales and Gross Margin
3. Projected Financial Statements and Cash Flow
4. Reinvestment of Earnings Model
5. Internal Rate of Return
6. Administrative Cost Center Budget
7. Budget Consolidation

Each of these examples is included on the diskette, and you can learn a great deal by simply running each of them, trying out the graphics mode, changing the PRINT system parameters, and even introducing syntax errors to observe the associated error codes that will result. Certainly, most of these examples can be easily adapted for use in establishing your own programs.

Sensitivity Analyses. Performing "WHAT IF...?" studies is the essence of modeling via computers. One means to accomplish this operation with PLAN80 is as follows. On-line entry of data values can be achieved by embedding a question mark in the DATA field for any particular variable. This feature, when combined with a recalculate command ("+" keystroke) and an appropriately placed INTERACTIVE statement, permits you to conduct a sensitivity analysis in a rather direct manner. The analysis can be terminated, saved, and printed out whenever you decide that this is the appropriate action to take.

Concluding Remarks

PLAN80 provides a powerful software package for numerical problem solving. Augmenting the printout capability is the graphic display (involving up to fourteen items per display) that can only be appreciated when examined firsthand by "hands on" testing in whatever areas of application you may have either needs or interests.

Expert. After you become well acquainted with PLAN80's conventions, notations, and mode selection keycodes, you will find yourself able to readily produce highly specialized programs to meet your applications requirements with but minimal referral to the PLAN80 manual. In fact, the Reference Card supplied with the manual will become a shorthand reminder of these conventions, etc. However, should you feel at any time that further clarification is required, the relevant pages in the manual for the particular PLAN80 operation or notation in question are conveniently indicated on the Reference Card.

Shortcoming. The principal shortcoming of PLAN80 (through version 2.2, at least) is the fact that graphics display is presently restricted to the screen of the CRT terminal. This is somewhat understandable since display-related parameters appear on the right hand side of the CRT screen, and these entries would not normally constitute a formalized hardcopy graphic representation. I would conclude, however, that this limitation will be removed (if it is not already in process or completed) by Business Planning Systems, Inc. once they receive sufficient feedback from otherwise satisfied users of PLAN80.

Economic Perspective. I have found that the more I studied financial planning, budgeting and control, economics, and engineering textbooks, the more I became convinced that PLAN80 would prove to be a boon to any company or individual requiring computer modeling at reasonable cost and under standalone system situations. Certainly, a reasonably sophisticated microcomputer system proves essential to obtain maximum benefits from this product, but it provides capabilities usually found in much more expensive packages.

Full Screen Program Editors: PMATE

Ward Christensen

PMATE (from Lifeboat Associates) is an immensely powerful, general purpose, full screen editor. Of the editors I have reviewed, it is the most customizable.

The PMATE command language is simply superb. I have never seen a microcomputer editor allowing such power, saving so much time on complicated editing functions. In addition to typical full screen capabilties, it provides a true editing oriented programming language. As such, I'm sure there are new things I will think of doing with it years from now. I am now working on a set of macros to do text printing, but more importantly, to gather an index of keywords from the document itself.

Mike Olfe is helping spread the word on PMATE with his column in *Lifelines*. One particularly useful macro Mike wrote will position you to a label in an ASM file; you simply place the cursor on a reference to the label in an instruction operand, and then invoke the macro. The macro then positions you to the label itself.

Subjective Evaluation

DOCUMENTATION: PMATE comes with an 80 page manual. The chapters are:

(1) introduction for beginners;
(2) basic concepts;
(3) advanced concepts and commands;
(4) complete command set;
(5) macro examples;
(6) configuration information;
(7) how to more extensively customize PMATE in assembler;
(8) a command reference appendix.

The documentation is quite complete. However, certain subtleties are only evident when you start using PMATE for yourself.

The manual has a nice five page command summary, but it doesn't include everything. An index would be helpful to find those commands not included in the summary. I find myself paging back and forth through the manual quite a bit, and have used a highlighting marker to help me find the more interesting sections.

SPEED: PMATE is adequately fast. If I didn't have WordMaster to compare it with, I probably wouldn't have noticed anything. However, in macros to search and insert characters, it is enormously slower than WordMaster. If I have a lot of text upon which to perform repetitive operations, such as stripping comments out of an ASM program, I use WordMaster. Also, PMATE lacks the all-important "repeat key" capability; however, its superb ability to be customized has allowed me to add a hack-repeat ability. More about that later.

ERGONOMICS: The default keyboard layout is "geometric". For example, the control characters G, H, Y, and B, are cursor left, right, up, and down. This is a half left-handed, half right-handed operation, and whether or not you'll like it is a matter of personal preference. However, the fact that you can customize makes the layout "anything you want".

I like to be able to scroll a document one line at a time, without moving the cursor. PMATE will scroll a line at a time, but only when the cursor has moved out of a range called *wander* – the number of lines away from the middle line of the screen. For example, if wander is 4, when you attempt to move the cursor more than 4 lines up from the middle of the screen, the screen scrolls down, and the cursor thus stays within wander.

I have set wander to 0, so the cursor stays in the middle of the screen. Thus cursor movement up or down causes the file to scroll one line, and the cursor stays at the middle line of the screen. Only when I was playing with PMATE on a 450 baud remote terminal did I set wander to a larger amount. This allowed me to edit quite a bit of a page without the remote system continually sending new top or bottom lines as I moved out of the wander limit.

CONFIGURABILITY: There is a configuration program to set up PMATE for a specific terminal. It is supported by configuration files for several common terminals, including memory mapped. If you do not have one of the standard terminals, you can edit the closest configuration file to customize it to your terminal. The only drawback is that you must key in the decimal or hex values for control keys as you assign them. How much more pleasant the configuration process would have been if "↑N" could have been put in to mean control-N, instead of having to do "20" or "14H".

In addition, if you don't mind using 8080 assembler, you can *really* make PMATE your personal "cup of tea". This also eliminates the step of keyboard assignment using the decimal or hex values for the keys.

NOTE: You are *not* programming *in* 8080 instructions, but only with define byte (DB) character strings, and define word (DW) addresses.

More than any other editor I have seen, PMATE allows you to add commands of your own, at several levels:

(1) PMATE has a text buffer called "T", and 10 auxiliary buffers numbered 0-9. You can place a command in any of the auxiliary buffers, and execute it by typing ".n" where "n" is the buffer number. The commands in one buffer may call those in another, like subroutines.

(2) You can create your own library of commonly used commands, and have them automatically patched into PMATE so that they are resident in a special "global macro area" every time you run PMATE.

With PMATE's flexibility you can even have several custom sets of "global macros", perhaps one for assembler programming, one for general text processing, etc. A command called QMC copies the current edit buffer to the global macro area. From there, the macros are easily executable within other commands. Global macros are named by a single character, and executed by ".x" where "x" is the macro name. There is even a special global command which is executed automatically when PMATE is run.

After placing your custom permanent macros in PMATE with the QMC command, the command XDfilename will "clone" the current copy of PMATE in memory (with its new global macros) to disk for future execution. To edit the global macros, a QMG command gets them into the current edit buffer.

(3) You can, with assembler programming, write entirely new "instant commands" - commands which are executed based upon keystrokes. They may be simple keystrokes: a control-B for example; or you may write a two-character sequence, so as to effectively double the number of different commands. They may be as obscure and specialized as you like. For example, I customized PMATE so that control-] followed by a dash puts a line of 60 dashes in the file. Another example: PMATE has a "kill line" control key, but it worked in such a way that if pressed while at the beginning of the line, it killed both the line and the carriage return at the end. If in the middle of the line, it was merely an "erase to end of line". I preferred to have two distinct keys - one to erase the entire line the cursor is on, no matter which column the cursor is in, and a second to erase to end of line, even if it is in column one. It was easy to add these functions.

EASY TO LEARN: The basic editing functions of PMATE are quite easy to learn. Like MINCE, however, it will take you many hours to try all the individual commands. The more time you spend with the book, the more you'll learn, and thus the more time you'll save. Also, the significant programmability means you will learn by sharing PMATE programming ideas with other PMATE users.

Objective Evaluation

Video-Related Criteria

FULL SCREEN: CP/M's ED has only a command mode. WordMaster has two distinct modes: a full screen mode, and a command mode in which the full screen "goes away".

With PMATE, you are *always* in full screen mode. The top line is a status line, showing the filename, buffer number, command numeric argument, and file line and column. Line two contains either the words OVERTYPE MODE or INSERT MODE, or your command.

There are keys to move the cursor one line up or down, one character left or right, or one word left or right. With customizing in assembler, you could add anything you want - something as obscure as a command to move half way from the column you are in to the left margin.

The default configuration does not have control keys to go to the beginning or end of a line, but they may be added in assembler. I customized a control key like WordMaster has: it moves to the front of the line if not there, or if there, moves to the end of the line.

SCROLLING: There are keys to scroll up or down a user-set number of lines. This may be a full screen, or perhaps two-thirds of a screen, whatever you like. A separate key goes to the top of the buffer [or if already there] to the bottom.

Terminal hardware line-insert and line-delete are supported, so small (say, 1-4 line) scrolls are very efficient on terminals that support this excellent feature. Without this support, WordMaster has to completely redraw the screen when scrolling backwards one line.

File scrolling is transparent to the user. The necessary scratch files are created on disk to allow full bi-directional scrolling of files bigger than memory. This applies only to the "T" (text) buffer.

You even have control over the bidirectional disk scrolling, with several commands. They refer to a "page" of data, which means a user-set number of lines, or a block of lines delimited by control-L (ASCII form feed). The commands are: XA - append next page of input; -XA - bring back a page already written to the output; XW - write a page to the temporary output file; XR - write one page out, and read a new one in. Any of the commands may have a number prefixing the "X", to indicate the number of pages.

These commands, along with the command "@M" which shows you how much free memory you currently have, allow you to scroll a large file through the text buffer, while keeping memory available for bringing in other files in other buffers.

INSERT: PMATE has a separate insert mode.

OVERTYPE: PMATE has an overtype mode.

UNDO-KEY: PMATE gets the prize here. Whatever you delete gets pushed onto a garbage stack, delimited from the previous deletions. On request, a control key "pops" the top of the stack. You can thus delete a character, then a line. Using the "pop" key would then bring back the line. Another "pop" brings back the single character. This is simply superb, and like the repeat key in MINCE, should set a standard for all text editors to follow.

REPEAT: *Nope!* "An editor without a repeat key is like a bicycle with flat tires." It can go from here to there, but it's not very fast and it's not much fun. I absolutely couldn't stand it.

I used the PMATE customizability to "hack up" a bearable substitute lacking repeat key. I specifically coded several two-keystroke commands, each of which start with control-W. The character after the control-W is one of the following: scroll up one page, scroll down one page, character left, right, up, or down, insert line, or character delete. Thus, even though no repeat key exists, I do have a way of doing "four of something" for at least a few of the more common cursor movements.

TEXT EDITING ABILITIES: PMATE has a fill mode, with word-wrap, so it is

(continued next page)

suitable for basic text editing. An "nnF" command sets formatted mode, with a right margin of nn. Like WordStar, it uses a special "soft carriage return", i.e. hex 8D, to indicate the "soft" end of a line. Thus a paragraph consists of "soft" terminated lines, and a final "hard" (hex 0D) carriage return.

PMATE also wraps at hyphens, but when you go out of formatted mode, any line that was wrapped at a hyphen becomes simply a very long line.

PMATE also has the ability to print, either a specific number of lines, or the entire buffer. It does not explicitly handle paging, or anything else. However, the crude print capability, combined with the powerful macro commands, could concievably be turned into a serviceable text processor. (As I said in the opening paragraphs, I'm working on it. The macros will be published in *Lifelines* if I ever complete them.)

Command-Related Criteria

MOVE: You may move to the top or bottom of the file (via "A" and "Z"), ahead or back by character (via "M" or "-M"), line (via "L" or "-L") or paragraph (via "P" or "-P"). In addition, "0L" moves to the front of the current line. A positive or negative number may precede these commands, or a "#" to indicate moving to a tagged location (tagging is done by a special control character).

DELETE: PMATE may delete characters (via "D") or kill lines (via "K"). A positive or negative number may precede the command, which indicates the direction and number of characters or lines to delete. "#" applies: "#K" will kill from the cursor, to the place marked with the tag control key.

INSERT: Arbitrary character strings may be inserted. The "I" command inserts the character string following it, up to a terminating ESC (which echoes as a "$"). To insert a single character, Ichar$ may be used, or, a number or character preceding the I: "xI would insert the character "x". (The leading" is part of the command – it is like a *function* that returns the value of the character following it).

Also, an arbitrary control character may be inserted into a file, by typing "↑" followed by a letter. To insert a real up arrow, you just press it twice. You may also insert control characters by using nnI to insert the value "nn" as a character in the file.

Character strings representing numeric values in any base may be inserted into the buffer, by the command nn\. "nn" may be a specific value, or a variable. Suppose that variable 6 contains the value 156, and that PMATE is in its default mode, decimal in and out. The command @6\ says to insert the value of variable 6 into the buffer. The buffer will then have, at the cursor position, "156". If you issued the command 16QO, which says make 16 the base for output, then @6\ would insert "9C" into the buffer. The command string "a1v1[@1\9iva1l]" would go to the top of the buffer, set variable one to 1, then number the line, followed by a tab, stopping when the bottom of the file is reached.

TYPE: Since PMATE always shows you a full screen of data, there is no *type* command, as was necessary in ED or WordMaster. Instead there is a command QR, which if placed in a command string, displays the full screen before continuing command execution. You may thus, by placing a QR command at appropriate places in your commands, watch the progress. On a memory mapped display, this works so fast, that you can literally write *movies* in PMATE; it automatically goes down a page, displays and moves, producing perhaps 8 frames per second on a 5MHZ system with an 80x24 screen! I use QR all the time on the memory mapped screen, but am more frugal with it on a 9600 baud terminal, since a screen update takes more like one or two seconds.

A QD command may be used to delay ("let you look") for a variable time before proceeding – i.e. QR5QD in a command will let you look at the screen a bit longer than just using QR.

FIND: PMATE can find strings, either staying within the current buffer contents, or scanning ahead with automatic disk buffering. It may also find backwards, and again be either limited to the buffer, or told to search the file already written out to disk.

CHANGE: Like find, CHANGE may be limited to the buffer, or allowed to go through the entire file, either forwards or backwards.

MOVE and COPY: These are implemented quite well: by marking one end of the block to be moved or copied, then going to the other end and typing a control key which moves it to buffer zero. If you want to copy, just press the control key to yank buffer zero back in, then reposition to where you eventually want the text, and yank it back again. To move, just omit the first yank. The only problem is that the marked block is invisible, with no good way to check where the mark is.

Other explicit commands support moving or copying text to one of the nine other buffers, as either a replace or an append: nBmC copies n lines to buffer m, overlaying whatever was there. nBmD appends n lines to buffer m. nBmM similarly moves, and nBmN similarly moves with append. You may also use tagging, but must issue the command #BmC for example, to indicate the tagged block is to be copied to buffer m.

COMMAND STRINGS: Nice. Superb. Unbelievable. Terrific. (Place your favorite superlative here).

PMATE command strings are so powerful, your time and imagination will likely be the only limits to what you do with them. PMATE even supports a trace mode to step through the execution of a complex command.

Let me start by comparing PMATE with the command language of ED. PMATE did not specifically attempt to mimic it, like WordMaster did. If you're familiar with ED, you may take a while to get used to C versus M to move a character, S versus C to substitute/change, B versus A to go to the top of the file, etc. ED's simple "M" macro repeat is not implemented. Instead, the commands to be repeated are placed in brackets: [and].

Still sounds like ED, right? Well, take the usual ED macro abilities, and add such goodies as conditional expressions (if/then), continue (repeat loop if condition met), break (break out of loop if condition met). What do I mean, "conditions"? Read on.

In PMATE, the macros may test for any of several conditions. Conditions

may be a single test, such as "@e" which tests the error flag; or they may be complex expressions involving "=", not "=", greater than, less than, ANDing, ORing, complementing, etc. (Whew!). What can you test? Any of about twenty-five things - what column the cursor is in, what the character under the cursor is, what line you are on, the amount of memory remaining in the buffer, even the value of the ASCII character struck to reply to a macro-imbedded prompt!

PMATE can also do arithmetic, and has 10 variables to use, as well as a stack to push and pop numeric values. Additionally, it can look at or change any byte of memory by placing its address in variable 9. @@ then refers to the byte pointed to by variable 9, and nQ! will store n into that byte. PMATE supports virtually any number base - so powerful that you may have it take octal on any number input, and display hex on any number output.

Macros can call other macros, either ones in another numbered buffer, or one of the permanent macros.

The macros are powerful, but what I have said doesn't adequately convey how they might be used. Let's take a specific example. In working on a Users' Guide for CBBS (The Computerized Bulletin Board System which Randy Suess and I developed), I wanted to publish the text of some of the HELP files. They are structured, such that in one line they ask if you want help on such-and-such. If you say no, then the text following the question, and any imbedded questions, are skipped. I wanted to show this very clearly by indenting, and by placing a column of "|" symbols, to line up the appropriate questions and their ends. This is very much like properly indenting a structured program, as you would in Pascal or C.

I first created a command which inserts "|bbb" (where "b" means a blank), based upon an indent count. I decided to place this command in buffer two. To do so, I typed: "b2e" which means Buffer two Edit. I then typed control-O, to go into overtype mode. I then typed the macro, which is, in "pseudo code":

Loop, repeating whatever the repeat count is, inserting one "|bbb" at the front of the line, then return.

I chose to keep the indent amount in variable one. Symbolically, it is referred to as @1. The macro in buffer two looked like:

```
@1[i|     $]
```

This consists of a repeat loop (delimited by an opening "[" and a closing "]"), executed the number of times contained in variable one. If variable one is zero nothing is done, i.e. the loop is executed zero times. Inside of the loop is the simple string "i" for insert, and the string itself, terminated by an ESC, which echoes as "$".

I then needed another command, which checked the first character in each line, to see if it was the start or end of a question. To see what I mean, here is a sample of what the original HELP file looked something like:

```
[Want to know how to scan?
To scan the summary, ...blah
blah...

You can also scan specific
fields in the summary.
]
[Want to know how?
You select the field: Date,
From, To, Subject, by its
first character...
]
[Want to use "and" and "or"?
If you want to...
```

I wanted this to look like:

```
[Want to know how to scan?
| To scan the summary, ...blah
| blah...
|
| You can also scan specific
| fields in the summary.
| [Want to know how?
| | You select the field:
| | Date, From, To,
| | Subject, by its
| | first character.
| ]
| [Want to use "and" and "or"?
| | If you want to...
| ]
|
```

The command necessary to do this follows (in pseudo code):

LOOP: (repeat this over and over)

Move down to the front of the next line.

If the character under the cursor is "[", then execute the command in buffer two to put the current indent, then add one to the indent amount, and "continue" the loop back at the top.

If the character under the cursor is a "]", i.e. the end of a question, subtract one from the indent amount.

Execute the command in buffer two, to set the indent for this line, whether it is an end ("]"), or just a text line.

As a PMATE command, that looks like:

```
[l@t="[{.2va1↑}
@t="][-1va1]
.2qr]
```

Let me split that out, to show you what each piece means.

'[' means begin a loop.

'l' means down one line (also implicitly moves to the front of the line).

'@t="[' is a PMATE conditional test. @t means the value of the character under the cursor, and "double quote char" means the value of that character. Thus this reads 'if the character under the cursor is "[", even though the second double quote is not shown.

'{': Following a conditional expression, the command contained in paired { and } or [and], is executed if the condition is met. Thus, the expression '.2va1↑' is executed if the character under the cursor is "[".

'.2' means to execute the command in buffer two. This puts the appropriate number of "|bbb" indents on the current line.

'va1' adds one to variable one. Had I wanted to add two, I would have had to explicitly give the two: 2va1.

'↑' means to "continue", i.e. go back to the top of the loop. PMATE makes some simplifications over what a high level language would do: the "top of the loop" is defined as simply the preceding "[" loop enclosing character. More specifically, loops enclosed in "{" and "}" are ignored. Thus "↑" looks

(continued next page)

back until it finds "[". (It skipped the "[because it was smart enough to recognize that as a character test, not a loop start.)

'}' closes the loop started by '{'.

'@t="]' tests if the character under the cursor is "]".

'[' starts the "then" part of the condition.

'-1va1' subtracts one from variable one, i.e. decrements the indent amount.

']' closes the "then" condition.

'.2' then executes buffer two, which places the appropriate number of "|bbb" in front of the line.

'qr' is optional, and shows the progress of the macro by updating the full screen as it now looks.

']' closes the initial whole-macro loop started by "[".

That's it! One example can hardly express the power of the language (for it *is* a language), but I hope you get the idea of how arbitrary a task it may be programmed to do.

MULTIPLE EDITS: PMATE, strictly speaking, edits only one file at a time, in its "T" (Text) buffer. However, you may switch to another buffer, and input another file. Management of the buffer space (i.e. bidirectional file scrolling) is only done in the text buffer, so the total size of files being used in other buffers is limited. However, PMATE allows paging pieces of a file in, where you specify the number of lines per page, or allow them to be delimited by control-L form feeds.

You can finish editing one file, then start editing another without leaving PMATE. This is nice, and saves time when editing several files.

File-Related Criteria

BACKUP: PMATE does the same as ED or WordMaster – after editing a file, the original is renamed to ".BAK"; a temporary file created by the editor is renamed to the original name of the file.

If you wish to eliminate the .BAK ability, and the file fits in memory, you can input the entire file (to any buffer), with an "XIfilename" command. After editing, you may XOfilename to save it. If you are saving it back under the same name, you must first explicitly delete the original by an XXfilename command.

SAVE: The XJ command is like the "H" command in ED or WordMaster: it finishes writing out the edited file to the work file, then renames the original file to .BAK, and renames the temporary file to the original filename, then starts editing that file again. It is exactly identical to leaving PMATE, then re-editing the same file.

QUIT: You can kill a buffer and discard the edit. You can then edit another file, the same file, or go back to CP/M.

READ: To insert a file from disk into the one you are editing, you simply "XIfilename". It must fit in the available memory. Optionally, you may set the page length (nQP) then "page" in the file one or more pages at a time: 20QP would set the page length to 20 lines. 1XIfilename would input the first page (20 lines) of the file. 1XI reads in the next page. 2XI would read in two pages. If you think you have room, XI would read in the rest of the file.

WRITE: You can write out the buffer to another file. To write out a marked piece of the file, first move the marked text to another buffer, then write it, then move it back.

DIRECTORY: PMATE nicely allows you to insert the directory of any disk into the file. One day during a big edit, PMATE told me my disk was full. I simply switched to buffer 1 (B1E), asked for a directory list of the entire disk (XLB:*.*) then began erasing (XXfilename) any files I no longer needed. Very nice!

OTHER: PMATE has a few features that don't fit the above categories. For example, horizontal scrolling of up to 250 characters can be executed. (Much longer lines can be handled, but you'll get to see only the first 250 chraracters). While intriguing at first, I found it to be unnecessary. If you have an application which is creating a document that must be printed more than 80 columns wide, it would be vital; but this feature is simply not needed for normal usage. (Hmmm, I can think of an exception. BDS-C programs, to maintain the nice structured indenting, often go wider than 80 columns.

Statistics

PMATE sells for $195, and is a Lifeboat product, the author being Phoenix Software Associates, Ltd. It takes about 20K of memory, depending upon how much you have added, and how big your global macro area is.

Room For Improvement

There is very little I would change. I have a few minor suggestions.

(1) A repeat key should be added.

(2) Some subtleties could be explained in the documentation, so you don't have to find them yourself. For example, conditions allow an if/ then/ else. It is implemented as: if condition [then expression][else expression]. It is not documented that the "][" pair of characters must be coded just like that; i.e. you can't put a space between the "]" and "[". Also, if you mean to *not* have an *else* condition, but want to start a loop (which starts with "[") then you *must* put a space or other character between the "]" that closed the "then" expression, and the "[" that begins the next loop.

(3) The handling of "-" in word-wrap is not useful, and confuses other text processing programs.

(4) The "W" command, to move a word, erroneously stops at ASCII tabs. It should stop only when a genuine "word" is encountered.

(5) For every character you type, the cursor jumps to the top right corner of the screen to update the column, then to the bottom right corner, to no imaginable purpose. Kind of "jumpy". I have switched from a reverse video block cursor (which I prefer) to a simple underline cursor, so the cursor "jumpiness" is not so distracting.

(6) In word-wrap mode, when the end of a sentence falls at the end of a

line, PMATE tosses the second space (which is technically correct to have) following the period. This will show up if the paragraph is reformed by editing or changing the right margin.

Conclusions

If you enjoy programming, you'll probably enjoy PMATE. The ability to write programs in your editor is really quite nice. I unequivocally recommend PMATE to any "hacker". You will love customizing it, adding your own commands, then writing macros for almost any task you can dream up. I see from recent Lifeboat ads that PMATE is now available for IBM P.C. DOS, SB-86, and MS-DOS.

Software Notes

Pseudo-Relocatable Subroutines

Gregory A. Knott

So you want to write an assembler subroutine, but you don't know how to place it in memory so MBASIC can get at it. Maybe you've been going through some of the frustration I did when I tried to do it, or maybe you have just given up, thinking the problem is beyond your level of expertise! But read on and we'll try to raise that level a little bit.

If you are one of the lucky ones (spelled "rich") who has a relocatable macro assembler, you have probably solved this problem already. But if you are like me (spelled "poor", and only able to buy new software by making grand promises to the spouse) you most likely have the assembler that was supplied with your original CP/M system. Well, it is possible to fake out Digital Research's ASM and end up with a program that is almost relocatable; all it takes is a little more care.

The Problem

One day, while minding my own business, I had a great idea for a subroutine. To implement it in MBASIC was practically impossible, so I decided to flex my assembly language abilities and code a routine that could be called from a BASIC program. For the purposes of this article, I have substituted a nonsense subroutine for the real one (obviously a no-nonsense one). This subroutine does nothing more than print a message to the CRT screen, but its simplicity will help to illustrate my point. Listing 1 shows the subroutine as it was originally written.

Let's quickly step through this routine. The ORG statement tells it to start at the beginning of the Transient Program Area (TPA). The MVI instruction places the CP/M Print Function code into Register C.

The LXI puts the address of the message we want printed in the Register Pair [D,E]. Now everything is set up so we can call the BDOS (at address 0005H) and CP/M will print along until it finds a Dollar Sign. After the printing is done, we end the program by returning to the Caller (the operating system - CP/M). Notice that the Message that we send also contains the codes for a line feed and a carriage return before it finds the Dollar Sign that delimits this message.

Pretty straightforward, right? Yep! As a matter of fact this program will run as a stand-alone COM file. But it isn't really a subroutine yet because this routine sits in the place where BASIC is loaded. To have BASIC call this routine it must reside in a memory location above BASIC but below the CP/M system FDOS.

How can this be a problem, when all we have to do is move this code up to a place where BASIC can use it? But there's more to the problem than that. Let's say that we want this code to reside at location B100 Hex. If we just placed the code there everything would work until we got to the LXI instruction. At this point the register pair [D,E] would contain the address of a message at 109H instead of B109H where the message really is now. If you followed that last sentence you understand that the ASM assembler assigns code addresses based on where they reside *relative to the TPA*.

The Solution That Wasn't

Let me help out by showing you how I tried to resolve this problem. I thought that by simply redefining the starting point of the program I could solve my problem and have a subroutine that could begin at B100H. In Listing 2 you can see how I changed the code.

As you see, the only change was to the ORG statement. Now the address of MSG is truly where it will be in memory. However, I was surprised when after it successfully compiled, it didn't LOAD the way I thought it would. Would you believe that this little .ASM file (that STAT tells me uses one record and 2k) turned into a monster after it went through the LOAD program? The .COM file that ensued was 353 records and took up 46k of my precious disk space. Besides, after trying to run it as is, I received the cryptic CP/M message BAD LOAD on my screen.

A little looking in the Digital Research manuals uncovered the problem. And I quote, "Further, the addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hex records are read. Thus, LOAD must be used only for creating CP/M standard "COM" files which operate in the TPA." Well, if I continued to create subroutines under this

(continued next page)

method I would quickly run out of disk space, so I would have to find another way.

The Solution That Was

The correct approach was to compile a program that was addressed relative to the beginning of the TPA - except for those areas that would need to be different if the program was moved. This would enable LOAD to deliver a normal sized .COM file and would allow for some relocatability. Relocatable assemblers mark the addresses that have to change; the linking loaders that accompany these assemblers change these addresses as the .REL files are loaded. Since ASM and LOAD do not perform this function the programmer has to do it for them. With the inclusion of two EQUate statements and the use of a "relocator" constant we can sufficiently fake out ASM and LOAD and get our relocatability. Because of our deception I like to think of these routines as "Pseudo-Relocatable". The final subroutine is in Listing 3.

The first EQUate statement is used to define the actual "relocated" start of the subroutine. All addresses that meet the "change" requirement due to relocation must be relative to this address. To get the amount of relocation required we subtract the subroutine start address from where we presently are in the program with our second EQUate statement. This equation here is LOC-$, which translates to B100H minus our current location (that's what the '$' means) of 100H. The result is a "relocator" constant that I have called "Z". If this relocator constant is added to all of the "change" addresses throughout the program they will assemble with the proper address locations. Look at the LXI statement now. We changed it by adding the relocator constant to it and now note that it assembled to the correct address of B109H that we were looking for. Voila! We now have what we wanted all along.

The Proof

To test this subroutine we can load it to the proper location by the use of DDT as follows:

```
A>DDT PRINTHI.COM
DDT VERSION 2.2
NEXT  PC
0180 0100
-M100,110,B100
-G0
```

What this does is start DDT with our pseudo-relocatable subroutine loaded at 100H. We move the subroutine (which happens to assemble into only 16 bytes) with the M command, saying move locations 100H through 110H to location B100H. Then we exit DDT with our command to G0 to the CP/M warm start location 0000H. We now have our subroutine loaded where we wanted it. It is now time to test it out. The following MBASIC program will work well enough:

```
10 ' PRINTEST.BAS -- TEST PRINTHI
20 PRINTHI%=&HB100
30 CALL PRINTHI%
40 END
```

When we load BASIC into the TPA we must remember to tell it not to overlay our subroutine. We do this and test out our program at the same time with the statment:

```
A>MBASIC PRINTHI /M:&HB100
```

This tells BASIC not to use any location higher than B100H and to execute the program "PRINTEST". After all this we find out that we were successful and our subroutine works.

(*Editor's Note:* Next month, look for Gregg Knott's loader, to put this subroutine into memory along with MBASIC.)

Listing 1

```
                ;       PRINTLO          MESSAGE PRINT SUBROUTINE
                ;
                ;       THIS IS THE SIMPLE ROUTINE
                ;
0100            PRINTLO ORG     0100H    ;BASE ADDRESS
0100 0E09               MVI     C,9      ;CP/M PRINT FUNCTION
0102 110901             LXI     D,MSG    ;[D,E] --> MESSAGE
0105 CD0500             CALL    0005H    ;LET BDOS PRINT IT
0108 C9                 RET              ;RETURN TO CALLER
                ;
0109 2A48492A   MSG     DB      '*HI*'
010D 0A0D24             DB      0AH,0DH,'$'
0110                    END     PRINTLO
```

Listing 2

```
                ;       PRINTRY          MESSAGE PRINT SUBROUTINE
                ;
                ;       ONLY CHANGED ORG'D LOCATION SO ROUTINE
                ;       COULD BE RUN AT MEMORY LOCATION B100
                ;
B100            PRINTRY ORG     0B100H   ;BASE ADDRESS
B100 0E09               MVI     C,9      ;CP/M PRINT FUNCTION
B102 1109B1             LXI     D,MSG    ;[D,E] --> MESSAGE
B105 CD0500             CALL    0005H    ;LET BDOS PRINT IT
B108 C9                 RET              ;RETURN TO CALLER
                ;
B109 2A48492A   MSG     DB      '*HI*'
B10D 0A0D24             DB      0AH,0DH,'$'
B110                    END     PRINTRY
```

Listing 3

```
                ;       PRINTHI          MESSAGE PRINT SUBROUTINE
                ;
                ;       ADDED EQUATES AND USED OFFSET ADDRESSING
                ;       TO MAKE "PSEUDO-RELOCATABLE" AT B100
                ;
0100            PRINTHI ORG     0100H    ;BASE ADDRESS
B100 =          LOC     EQU     0B100H   ;ROUTINE ADDRESS
B000 =          Z       EQU     LOC-$    ;ADDRESS "RELOCATOR"
0100 0E09               MVI     C,9      ;CP/M PRINT FUNCTION
0102 1109B1             LXI     D,MSG+Z  ;[D,E] --> MESSAGE
0105 CD0500             CALL    0005H    ;LET BDOS PRINT IT
0108 C9                 RET              ;RETURN TO CALLER
                ;
0109 2A48492A   MSG     DB      '*HI*'
010D 0A0D24             DB      0AH,0DH,'$'
0110                    END     PRINTHI
```

The April issue was placed into the mail on March 25th. 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 April 1982, into the mail around April 26th. We will print each month the date of the previous issue's mailing and would appreciate your help in tracking the deliveries.

Notice

CPMUG Volume 80, (continued from page 22)

80.16	2K	SPEAR.STB	Statistics program
80.17	6K	SPELL.STB	Part of spelling program
80.18	10K	STAT.STB	Statistics program
80.19	20K	TRADE.STB	Game prog with no GOTOs!
80.20	4K	TRANSFER.STB	Part of spelling program
80.21	24K	TREK.STB	Yes, Startrek
80.22	16K	TRK-HELP.STB	More Startrek stuff
80.23	6K	UTILITY.STB	"Unnum" and renum BASIC progs
80.24	40K	WORDLIST.TXT	Part of spelling program
80.25	2K	X2.STB	Statistics program
80.26	2K	ZIPSORT.STB	Part of mail list system

Prepared by Ward Christensen 04/82

editor to correct any misspelled words. Some other limitations of the program are that words longer than 15 characters are automatically put in the check file since the dictionary won't store them properly; proper nouns may be stored in the dictionary capitalized, but the spelling checker does not check the text for capitalization.

DATE.STB, MMENU.STB, REC-EDIT.STB, REC-PRN.-STB, ZIPSORT.STB and BACKUP.STB are all parts of a mailing list program. It allows names to be entered, modified and deleted. Printouts may be made of all the entered data, mailing labels sorted by zip code or alphabetically and in a membership list format. The program should be easy to customize.

CONV-ASC.STB and **CONV-BAS.STB** were written to help convert Microsoft and CBASIC programs into structured BASIC programs. I have used them to help convert most of the CPMUG games. Some limitations are that String functions and disk I/O usually still need additional work. Also CBASIC programs that do not have sequential line numbers or line numbers that are too close together will not convert. I found that about two-thirds of the programs will run properly after being "put through" the conversion programs; the others require additional modification.

STAT.STB, X2.STB and SPEAR.STB are all statistics programs using standard algorithms.

TREK.STB and TRK-HELP.STB are part of a new Startrek program that I wrote that is based on every other Startrek program that I have ever used.

GRADER.STB uses the algorithm by Donald Goodman and Sandra Schwab published in *Creative Computing* to calculate the Fog Index and the Flesch readability scale. The program itself has been rewritten to operate on text files rather than having someone type in text manually in a special format.

SORTS.STB was inspired by the two-part article in August and September 1981 *Interface Age* by Gene Cotton comparing different sorting routines. It lets the users compare different standard sorting routines done on an array of random numbers. The procedures may also be used in other programs.

UTILITY.STB is a program that will "strip" the line numbers off of a Cromemco BASIC program and later add them again. I have found it useful to use in combination with a text editor if blocks of a program need to be rearranged. Unfortunately it does not take GOTOs into account. If you need to alter a program that uses GOTOs I would suggest using labels instead of numbers.

TRADE.STB is a Structured BASIC version of an old CPMUG game from volume 21. The game has been rewritten to follow a structured format. There is not one Goto in the entire program!

PRN-TEST.STB is a simple printer testing program.

David E. Trachtenberg

(continued from page 16)

gram command line) to the FIFO. This is in fact a network transfer, using the high-speed Z80 block moves of our network drivers, and occurs in somewhat less than one character time. Since the destination FIFO is RAM-resident, the error-return from the master processor is done with no disk access necessary, yielding an extremely high rate of transfer. As a result we now have the ability to record incoming data from our serial device, at a relatively high speed (about 1800 baud), with a capacity limited only by the capacity of the disk.

System Security

TURBODOS may be configured at system generation time so that users are required to log in before using the system. Two utilities are provided to support this feature, LOGON and LOGOUT.

The LOGON program must exist in the logged-out user area of the disk (an area specified at system generation time), and, for proper security, should be the only executable file in this area on any drive. LOGON prompts for a user I.D. and password, which it validates against a system file named "USERID.SYS". USERID.SYS is created with a text editor, and contains entries of the form

USERID,[PASSWORD],USERNO["P"],[DRIVE]

where items in brackets are optional. The "P" option specifies a "privileged" user, who may, among other things, change user areas, and reset slaves. The P option also sets a flag in the operating system that can be queried via a system call, to allow a program to determine whether a user is privileged.

If the LOGON program finds the file "SYSLOG.SYS" in the logged-out user area, it will record usage information (name, date, time, etc) in a random-access disk file.

The LOGOFF utility changes the current user area to the logged-out user area, and posts a "logged-out" entry to USERID.SYS (if it exists).

Logon and logoff functions are also supported by BDOS calls. In fact, any system process that requires access to privileged functions (such as set user, etc.) must log in via this BDOS call, whether or not it is associated with a console.

System Utilities

Beside the utilities already mentioned, there are a number of other utilities provided, which I'll describe now.

COPY. This is the TURBODOS replacement for CP/M's PIP.COM. It lacks the text formatting capabilities of PIP, but does provide a number of new features. COPY will accept wildcard filename specifications (as do most TURBODOS utilities), and allows command-line options to specify source and destination user number, select non-archived files only (TURBODOS always resets the "archived" attributes of files when they are written to), delete each file from the source disk/user after it is copied, and allow a destination disk to be changed if it becomes full. Another feature of the copy program is its ability to rename files (when wildcards are specified) as they are copied. For example, the command "COPY *.BAS B:*.001" will copy all the files with type BAS to the B drive, and rename them with type "001" at the destination.

AUTOLOAD. This utility writes its command tail to a file named "AUTOSTRT.AUT" in a special format that permits the command line to be executed at initial cold-start and/or warm start. By renaming AUTOSTRT.AUT to WARMSTRT.AUT, the user will be forced to execute the specified command line at each warm start (which occurs whenever a transient program completes execution). The corresponding action for initial system cold start is done by naming the file COLDSTRT.AUT.

DIR. This is similar to the resident DIR command of CP/M. It prints an alphabetically sorted listing, displays the amount of free space on the disk, and reports the size of each file. Also DIR displays the disk label, which may be created using the LABEL utility.

RENAME. This utility renames files. It's similar to REN of CP/M, with a couple of major differences. First, the syntax is reversed; the CP/M syntax is "REN NEWFILE.NAM=OLDFILE.NAM" while RENAME syntax is "RENAME OLDFILE.NAM NEWFILE.NAM". Secondly, RENAME allows wildcard file specifications on *both* the source and destination names. When used in this way, wildcard characters used in the NEWFILE argument indicate that the corresponding characters of each old file name are to be used in renaming each file. For example, the command "RENAME *.BAS *.OLD" will rename all files of type BAS to type OLD.

SET, SHOW. These utilities are used to manipulate file attributes. The SET utility allows any or all supported file attributes to be set using a mnemonic letter. SHOW displays the attributes. Both allow wild-card file specifications.

In addition, utilities are provided for setting and displaying time and date, dynamically altering disk buffering parameters, displaying drive characteristics, initializing and copying disks, resetting slaves, typing out ASCII files, testing disks, and dumping files in combined hex and ASCII format.

Pitfalls

As you've probably guessed, I'm very enthusiastic about TURBODOS; I've used it daily for several months, and am impressed with its power and reliability. Yet there are a few potential problems that an informed buyer should beware of.

CP/M Incompatibilities

TURBODOS does not maintain an in-memory disk allocation map, as CP/M does. For that reason, programs using 27 ("return disk allocation address") will not run correctly under TURBODOS. The only programs I know of that use this function are the CP/M STAT utility and the public domain "SD.COM", both of which use this function to report free disk space (both of which also fail miserably when executed under TURBODOS). The TURBODOS function 27 *does* return the number of free disk blocks, but programs must be modified at the source level to take advantage of this. I was able to make the necessary changes to SD in a few hours.

A more serious problem occurs with programs that make use of the CP/M

disk parameter block to determine the physical disk characteristics. TURBODOS does not support the CP/M disk parameter block, and the associated system call (number 31) returns the TURBODOS-compatible disk information (again, a competent systems hacker *can* modify a program at the source level to use the TURBODOS format). The programs I know of that do not work properly because of this problem are the public domain DU, SAPX and FINDBAD disk utilities, and the commercial diagnostics package RECLAIM.

Another potential problem occurs when TURBODOS disk drivers are set up with reserved tracks (I do this to maintain CP/M compatibility). Although a complete BIOS jump table is simulated by the system, the "set track" subroutine *cannot* set a track within the reserved track area. This caused a particularly frustrating problem with my own IBM disk-reader program; those tracks within the CP/M reserved track area were just not available. The only solution was to add two "pseudo-drives" ("G" and "H") to the system, which are, in fact, the same physical drives as our floppy "E" and "F", but with no reserved tracks.

The only commercial package that I know of that would not function because of this reserved track problem is the REFORMATTER package, a disk utility for IBM-format disks.

Operating System Overhead

A problem related to operating system overhead occurs with programs (such as WordStar) that make frequent BIOS calls to test console status. This test, under CP/M, involves perhaps a half-dozen machine instructions. TURBODOS, however, converts these calls to system calls, which carry the overhead of hundreds of machine instructions. The result is that such programs run much more slowly under TURBODOS.

In the particular case of WordStar, the rate of console status checking can be adjusted by the user (this is detailed in the WordStar manual). A documentation supplement for TURBODOS from Software 2000 points this out, and provides details for adjusting WordStar to fit TURBODOS.

The Future of TURBODOS

I recently spoke with one of the two TURBODOS architects, Ronald Raikes of Software 2000, about his firm's plans for the continuing development of the operating system.

A full master-to-master networking system, supporting slaves with local disk storage, should be available soon. This will be revision 1.2 of TURBODOS, and is currently in the final testing stages. Additionally, many of the utilities will have enhanced features.

Another major release of the operating system (revision 1.3) is planned for later this year. This version will support bank-selectable memory, and an additional console per CPU. Mr. Raikes stressed that while he *will* support multiple consoles, a more important goal is based around a single console with two banks of memory, using one bank of memory for a "deluxe" operating system, and allowing a large (63 Kbyte) transient program area in the other.

Release 1.4 of TURBODOS will support a nested directory structure, similar to that of UNIX.

Technical Information

The following section contains information that may be useful to programmers who want to implement the operating system, or write transient programs running under it.

Extended BDOS Calls

In addition to the system calls provided with CP/M 2.2, a number of other useful functions are designed into TURBODOS. Aside from those I've already mentioned, system calls are available to set and return the date and time (using a binary rather than BCD format), set and return disk buffer parameters (both the size and the number of buffers may be specified), disable or enable the AUTOLOAD feature, load a program into memory (starting at the current DMA address; this allows overlays to be directly supported by the operating system), rebuild the disk allocation map, flush the disk buffers, lock and unlock drives, and lock and unlock printers.

The following system calls exist, and are listed in the manual, but are not available for use without help from Software 2000: reset network, send message to network, receive message from network, allocate memory segment, de-allocate memory segment, send and receive inter-process message, delay process, create process, and terminate process.

Disk Allocation

The disk allocation map is maintained on the disk, providing a high degree of file integrity when compared to systems that maintain this information in memory. I've found that, under TURBODOS, if I forget to close a file (or, if my program dies before it reaches the file close subroutine) no data is lost.

Dispatching

TURBODOS employs an interrupt-driven dispatcher to perform task-swapping in real-time. As far as I can tell, all processes receive the same priority, and are switched in a round-robin fashion. The dispatcher may be accessed from within any user-supplied modules via the WAIT and SIGNAL routines (also known as P and V in concurrent programming). These routines may be called by name within driver code (they must be declared as externals), and are passed EVENT SEMAPHORES as arguments. When WAIT is called, the semaphore counter is decremented, and, if the count is negative, the calling process is removed from the ready list until some other process calls SIGNAL with the same semaphore. Generally, the dispatcher overhead is such that a significant speed improvement is realized in slave CPUs by omitting the dispatcher altogether (this is done by selecting a non-multitasking STDSLAVE at system generation time).

Device Drivers

Complete programming specifications are provided in the documentation set. I've found them to be relatively complete, and have had no trouble writing device drivers for such things as an IBM card punch, a NEC printer, a PMMI

(continued next page)

modem, DJ2D floppy disks, and a Godbout System Support I clock/interrupt system, entirely from the detailed specifications (along with a little help from the supplied sample driver listings).

Interesting but as yet undocumented is the capability of the system to make recursive system calls from within itself (at the driver level). A recursive entry point, "XECFCN", is available for this purpose, and can be used to access any of the system calls. It should be possible, using this entry point, to perform disk reads and writes from within console drivers. Extreme care must be taken to avoid such problems as mutual lockout (calling disk functions within disk drivers at certain critical points will effectively block a process because of built-in mutual exclusion mechanisms) and changing the calling programs DMA address and user number, without restoring them.

Background Processes

User-written background processes are supported under TURBODOS, but the documentation as yet provides little information in their use. Since I had a need for using a background process, I had to contact my dealer (Musys Corp.) He in turn referred me to Software 2000, who provided all the information I needed.

Background processes must be linked into the operating system at system generation time. An explicit call to the operating system "create process" function must be made in the user-changeable hardware initialization module, with the process entry point passed in the DE register pair. The new process will then begin life with a 48-level stack, and its own set of registers, which are maintained between task swaps by the dispatcher.

All processes have access to the system as if they were the only running program in the system. System information such as the current DMA address, default drive, and current user number is maintained within the process descriptor of each process, and can be freely changed without affecting any other process in the system. System calls must be made by calling the global "OSNTRY", following the parameter passing conventions of "normal" transient programs. The only restriction is that all processes must *not* make use of the X index register.[8]

References

[1]Printer spooling generally refers to the technique of sending printer output to a disk file (or other storage medium) for later "despooling"; it usually takes the form of a background process in the master processor that selects the spooled files, one by one, for output to the printer. This technique allows orderly access to system printers, and is usually transparent to the user (despooling takes place concurrently with other system operations).

[2]File lockout refers to the ability of an operating system to allow exclusive access to a file for certain programs (or users), and deny all others. This is usually necessary when a program is updating a random record; until the update is complete, all other users must be denied access, to avoid reading inconsistent data.

[3]June, 1981 issue.

[4]CP/NET, a networking system for MP/M, was available at the time from Digital Research, but the preliminary system information we were able to obtain at the time was sketchy; we could not get enough details to allow a reasonable evaluation.

[5]I should note here that Software 2000 requests end users to report problems to the dealer from whom they purchased the system (MuSys in our case); if for some reason the dealer cannot resolve the problem, it then is the dealer's responsibility to put the end user in contact with Software 2000.

[6]There is a fundamental difference between TURBODOS and MP/M with respect to the "pass command line" system call that should be noted here. While MP/M immediately executes the passed command line, TURBODOS "stacks" the line, and will not execute it until the calling program terminates.

[7]A similar capability now exists for CP/M, using my SUPERSUB program (see the January, 1982 issue of *Lifelines*).

[8]This is a system-wide requirement in all drivers and processes. Transient programs, however, are controlled by a module called "LOCUSR", which supports "private" X-register usage, as well as a simulated BIOS jump table.

Opinion Letters

To Set The Record Straight

March 13, 1982

Lifelines:

Just a comment on Mr. Harnell's letter to me. I am not deserving of all the credit he gives me. It is other people who have taken my programs, and made them more useful: Mark Zeiger and his version 7 of MODEM, Keith Petersen, who wrote XMODEM because I was unwilling to admit people would have trouble remembering "MODEM SQ filename" (if they forgot the "Q" while running on a remote CP/M system, the transfer won't work), and Ron Fowler, who turned my single-density-only disk utility "DU" into one that can handle virtually any 1.4 or 2.2 disk format or density. (See DUU on CPMUG Volume 78).

Just wanted to set the record straight.

Ward Christensen

A Reply

March 8, 1982

Dear Editor,

Thank you for letting Mr. Jim Mills review our disk utilities DDUMP and DTEST in the February (82) issue of *Lifelines*.

Mr. Mills is quite right in saying that our utilities will probably not do much more than some readily available public domain disk utilities for CP/M (e.g., from CPMUG), but that is only true for 8" single sided, single density (SS/SD) diskettes (as Mr. Mills uses) or for a very limited number of particular disk controllers. If you use some other disk format, the CPMUG or other public domain utilities may not be of much use to you. We use Zenith/Heath CP/M, which supports no less than 10 different disk formats.

I was surprised to see that Mr. Mills totally missed this MAIN point about our disk utilities; namely that they work with most diskette types and formats — and that they are not limited to some special disk controller, or 8" SS/SD disks only!

This is also the reason why we wrote these utilities. After having searched around for general disk utilities which could be used with all our various disk formats — without finding any at all — we decided to write our own. We have now released a complete package of five well documented, user-friendly disk utilities. We also plan to release new versions which will also work with any hard disk format.

The reviewer of our software seems to be heavily biased against any utility for CP/M offered for sale — on the alleged ground that a similar utility can always be found in the public domain.

Regards,
Terje Bølstad
Elektrokonsult AS
Konnerudgaten 3,
N-3000 Drammen, Norway.

A Plea

March 19, 1982

Dear Sir:

I'm writing to ask your help in finding an S-100 interface card to a CalComp model 114 disk drive. These drives (which originally cost about $20,000) are mechanically very rugged and the removable disk packs can each store about 30 MBytes. This drive, unfortunately, does not have an SMD interface.

I thought I had found an electronic development company on the West Coast to build the interface but the press of other business has shelved the project for two years with no completion date in sight. Hence I'm again looking for a supplier.

There were thousands of these drives built and they are now being replaced by newer technology so it seems there is a good market for an S-100 card. I have complete technical information (Manual and schematics) which I'll make available if needed.

Sincerely,
Ron Tipton
The Systems Shoppe
Greenwood, Missouri

Correction

March 17, 1982

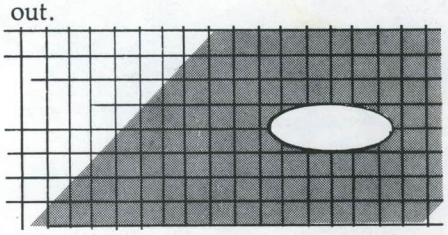
Dear Sir:

In "Tips and Techniques" of Volume 2, number 10, you explain a method interrupt-enable flag testing. Due to a bug in the Z80, this method will not work. The LD A,R or LD A,I instructions are supposed to set or clear the parity flag to reflect the state of the CPU's interrupt enable flag. If interrupts are enabled, and an interrupt occurs while the LD A,R (or LD A,I) is executing, the parity flag is not set properly. The current Zilog and Mostek manuals do not mention the problem (I wonder if future manuals will?). It is mentioned in Dr. Dobbs Journal, Number 58, August 1981 on page 40.

This bug will cause random system crashes which may be impossible to troubleshoot. We discovered the problem while debugging some code which contained the LD, A,I instructions running in a tight loop. The system crashed every few seconds, and the problem was finally traced to the LD A,I.

Sincerely,
Jim Freeman
NH Research, Incorporated

Editor's Note: We also thank Wayne Farmer of San Diego for pointing this out.



A Spelling Program (continued from page 11)

8080 Compatible Code for Spelling Correction

I have included the following listing of 8080 compatible code for doing spelling correction. It is abstracted from the source code for The Stiff Upper Lisp. It is presented here for those readers who may be interested in experimenting with spelling correction or in improving on the algorithm. The code presented here is for comparing an unknown word to one candidate word. If they match, the zero flag is set. I would be eager to hear from any readers who produce improved spelling correctors.

```
spell2 add a,b ;grace-difference
 jp nc,spell3 ;brif no grace left
 jp z,spell3
 ld b,a ;new grace to b
spll inc hl
spell5 call compchar
 ld a,b ;check grace
 and a
 jp z,spell3 ;brif no grace left
 dec c ;dec length
 jp z,spell4 ;brif words matched
 inc de
 inc hl
 jp spell5
spell4 pop de
 pop bc
 xor a ;set z flag
 ret
spell3 pop de
 pop bc
 or 1 ;set nz flag on no match
 ret
;
;compchar is the function that does the real work in the spell
;routine.
;
;compchar compares 1 char from 2 strings for approximate equality
;the following discrepencies are ignored:
;       2 chars transposed -- apepnd
;       repeated char -- appennd
;       omitted char -- appnd
;       extra char -- appexnd
;other discrepencies count against a match.  if too many discrepencies
;are found, relative to the length of the word, the strings do not match.
;c - length, b - grace factor, de - ptr to word, hl - ptr to known word
;
compchar ld a,(de)
 cp (hl)
 ret z
;transpose
 inc de
 ld a,(de)
 dec de
 cp (hl)
 jp nz,cc1 ;brif not a transpose
 inc hl
 ld a,(de)
 cp (hl)
 dec hl
 jp nz,cc3 ;brif not transpose, must be extra
 inc hl ;advance 1
 inc de
 ret
cc3 inc de
 dec b
 dec c
 ret nz
 inc c
;
;spell
;subr (spell <misspelling> <correctspelling>)
spell call ispell
 ld hl,nil
 ret nz
 ld h,d
 ld l,e
 ret
;
;internal version of spell
;
ispell push bc
 push de
 push hl
 ld b,grace ;grace
 call cdr ;get ptr to pname
 ld a,(hl) ;length to a
 ld c,a ;length to c
 cp 9 ;decrease grace for short words
 jp nc,spell1
 dec b
 cp 5
 jp nc,spell1
 dec b
spell1 inc hl ;ptr to string to de
 ex de,hl
 call spell8 ;compare one known word to word
 jp nz,spell7 ;brif no match
 pop hl
 pop de
 pop bc
 xor a ;set z flag
 ret
spell7 pop hl
 pop de
 pop bc
 or 255 ;set nz flag
 ret
;
;spell8 compares two words, word pname in de, known word atom in hl
;b - grace, c - length of word, both unchanged
;
;the main function of spell8 is to manage the values of the grace
;and word length, and to call compchar.
;
spell8 push bc ;save grace and length
 push de
 call cdr ;pname of known word
 ld a,(hl) ;length of known word
 sub c ;known length-unknown length
 jp z,spl1
 jp c,spell2 ;get -(abs) of difference
 cpl
 inc a
 ret
cc1 dec hl ;stutter
 ld a,(de)
 cp (hl)
 inc hl
 jp nz,cc2 ;brif not stutter
 dec hl
 dec c
 ret nz
 inc c
 ret
cc2 inc hl ;omitted
 ld a,(de)
 cp (hl)
 dec hl
 jp nz,cc4 ;brif not omit
 inc hl
 ret
cc4 dec b ;unknown error type
 ret
```

Misspellings As Run Through SPELL

Here are the most commonly misspelled words as reported in Thomas C. Pollack and William D. Baker, *The University Spelling Book*, Englewood Cliffs, New Jersey: Prentice-Hall, 1955. The misspellings are my own.

Misspelling	Correct Spelling	Output of SPELL
accomodate	accommodate →	accommodate
achievement	acheivement →	acheivement
aquire	acquire →	acquire
amog	among →	nil
aparent	apparent →	apparent
apparant	apparent →	apparent
arguement	argument →	nil
argueing	arguing →	nil
beleif	belief →	belief
beleive	believe →	believe
benificial	beneficial →	beneficial
benefitted	benefited →	benefited
catagory	category →	category
comming	coming →	nil
comparitive	comparative →	comparative
conscous	conscious →	conscious
contriversy	controversy →	controversy
contriversial	controversial →	controversial
definately	definitely →	definitely
definitly	definitely →	definitely
dephine	define →	nil
discribe	describe →	describe
description	description →	description
disasterous	disastrous →	disastrous
embarass	embarrass →	embarrass
envirnment	environment →	environment
exagerate	exaggerate →	exaggerate
existance	existence →↑	existence
existant	existent →	existent
expereince	experience →	experience
explaination	explanation →	explanation
fasinate	fascinate →	fascinate
hieght	height →	height
intrest	interest →	nil
lead	led →	nil
loose	lose →	nil
loosing	losing →	nil
marrage	marriage →	marriage
mear	mere →	nil

nesessary	necessary →	necessary
ocasion	occasion →	occasion
occured	occurred →	occurred
occured	occurred →	occurred
occurence	occurrence →	occurrence
occurrance	occurrence →	occurrence
occurance	occurrence →	occurrence
oppinion	opinion →	nil
oportunity	opportunity →	opportunity
pade	paid →	nil
perticular	particular →	particular
particuler	particular →	particular
perticuler	particular →	particular
preformence	performance →	performance
personall	personal →	personal
personel	personnel →	personnel
personnal	personnel →	personnel
personal	personnel →	nil
posession	possession →	possession
possesion	possession →	possession
posesion	possession →	nil
posible	possible →	possible
possable	possible →	possible
posable	possible →	nil
practicle	practical →	practical
preceed	precede →	precede
pregudice	prejudice →	prejudice
prepar	prepare →	prepare
prevelent	prevalent →	prevalent
principle	principal →	principal
principal	principle →	principle
privelege	privilege →	privilege
probaly	probably →	probably
procede	proceed →	proceed
proceedure	procedure →	procedure
proffessor	professor →	professor
proffession	profession →	profession
prominant	prominent →	prominent
persue	pursue →	pursue
quite	quiet →	quiet
recieve	receive →	receive
recieving	receiving →	receiving
recomend	recommend →	recommend
refering	referring →	referring
repitition	repetition →	repetition
rythm	rhythm →	rhythm
sence	sense →	sense
seperate	separate →	separate
seperation	separation →	separation
shinning	shining →	nil
similer	similar →	similar
studing	studying →	studying
suceed	succeed →	succeed
suprise	surprise →	nil
tecknique	technique →	technique
thier	their →	their
through	thorough →	nil
transfered	transferred →	transferred
unecessary	unnecessary →	unnecessary
unnesessary	unnecessary →	unnecessary
unesessary	unnecessary →	unnecessary
villin	villain →	villain
writting	writing →	nil

count = 104 matches = 87

Comparison of Misspellings with Close Words

In an attempt to determine if the algorithm for the SPELL function is too forgiving, the four words in the dictionary that were closest to the target word were compared with misspellings of that word. Misspellings were selected from the other spelling list; every fifth misspelling was used. This test may be somewhat misleading in that words that are not alphabetically close to another word may be matched by SPELL.

Misspelling	Close Word	Output of SPELL
accomodate	acclivity →	nil
accomodate	accolade →	nil
accomodate	accommodating →	nil
accomodate	accommodation →	nil
aparent	apparatus →	nil
aparent	apparel →	nil
aparent	apparition →	nil
aparent	apparitor →	nil
beleive	belief →	nil
beleive	believable →	nil
beleive	belike →	nil
beleive	belittle →	nil
comparitive	comparable →	nil
comparitive	comparatist →	nil
comparitive	comparativist →	nil
comparitive	comparator →	nil
definitly	definiendum →	nil
definitly	definiens →	nil
definitly	definition →	nil
definitly	definitive →	nil
embarass	embargo →	nil
embarass	embark →	nil
embarass	embarrassedly →	nil
embarass	embarrassingly →	nil
lead	lectotype →	nil
lead	lecture →	nil
lead	leda →	nil
lead	lederhosen →	nil
nesessary	nebulous →	nil
nesessary	necessarily →	nil
nesessary	necessarianism →	nil
nesessary	necessitate →	nil
occurrance	occupy →	nil
occurrance	occur →	nil
occurrance	occurrent →	nil
occurrance	ocean →	nil
perticular	participle →	nil
perticular	particle →	nil
perticular	particularism →	nil
perticular	particularity →	nil
preformence	perforated →	nil
preformence	perforation →	nil
preformence	performative →	nil
preformence	performer →	nil
posession	possessed →	nil
posession	possessed →	nil
posession	possessive →	nil
posession	possessory →	nil
posable	posset →	nil
posable	possibility →	nil
posable	possum →	nil
posable	post →	nil
prevelent	prevailing →	nil
prevelent	prevalence →	nil
prevelent	prevaricate →	nil
prevelent	prevenance →	nil
procede	procedural →	nil
procede	procedure →	nil
procede	proceeding →	nil
procede	proceeds →	nil
persue	purslane →	nil
persue	pursuance →	nil
persue	pursuit →	nil
persue	pursuivant →	nil
seperation	separable →	nil
seperation	separate →	nil
seperation	separationist →	nil
seperation	separatism →	nil
suprise	surprint →	nil
suprise	surprisal →	nil
suprise	surprising →	nil
suprise	surra →	nil
through	thorny →	nil
through	thoron →	nil
through	thoroughbass →	nil
through	thoroughbrace →	nil
villin	villager →	nil
villin	villiagery →	nil
villin	villainess →	nil
villin	villainous →	nil

count = 88 matches = 3

Product Status Reports

New Products

These products are available from their authors, computer stores, software distributors and software publishers.

ASCOM
Dynamic Microprocessor Associates, Inc.

The Asynchronous Communication Control Program is a communications facility for microcomputers running under CP/M-80 (or a CP/M-80 compatible operating system), allowing data transfer into and out of a serial port. Communications parameters such as baud rate and parity are set at the command level, and ASCOM permits file DIRectory, REName, DELete and TYPE without exiting to the system. ASCOM communicates with any machine capable of asynchronous communications.

Ma[illegible] features include a conversational mode option, interactive or batched command processing, simultaneous print operation, receive and transmit timeout, software parity check, several protocols, and a special help feature.

A 16K 8080 or Z80 microcomputer and 4K RAM are required. A knowledge of assembly language may be necessary for non-standard installation.

EM80/86
Dynamic Microprocessor Associates, Inc.

This emulator permits unmodified user programs written for CP/M-80 to be executed on an 8086/8088 running under IBM Personal Computer DOS (Seattle Computer Products' 86-DOS, Lifeboat Associates' SB-86, Microsoft's MSDOS) or CP/M-86.

8080 object files are transferred to the target 8086 system by entering the name of the program object file and the optional fields (those normally entered when the program is executed on an 8080 system).

I/O runs at operating system speed, so that programs mostly interacting with system peripherals will run as fast as on the original 8080 system. However, compute bound programs (or number crunchers) will execute much more slowly.

EM80/86 requires 4K of memory.

The Formula
Dynamic Microprocessor Associates, Inc.

This application development tool is intended to combine a data base manager, a word processor, and a compiler language, providing a "system language" for business applications. Various features simplify the creation of program-like modules; free format reports are generated from a visual description of the report, and file maintenance and data entry routines can be designed automatically using a description of the data.

The Formula contains an Indexed Sequential Access Method for data retrieval and executes object code modules. It also features a customizable full screen editor, text highlighting and underlining, user-designed menus, multiple access keys to the data, reporting and updating with conditional testing and algorithmic calculations, DATE conversions for numeric calculations. File record layouts and report descriptions can be generated.

The package includes a General Accounting System consisting of General Ledger, Accounts Payable, and Accounts Receivable. Guidelines are given on developing inventory and mailing software.

The Formula runs with a Z80 or 8080 CPU, CP/M-80 compatible operating systems.

Programmer's Apprentice
The Software Group

This program generator is designed to create fully debugged and commented application programs in MBASIC source code. Applications programs are menu-driven, and additional operator prompts can be provided. Program linking can result in application systems to run on a variety of terminals or memory-mapped desk top computers. Recursive data entry techniques are intended to speed program creation and data entry. Field definitions can be specified to an extent that will help eliminate input errors in data entry.

Screens, report formats, and masks can be designed interactively; report and input screens are available either as MBASIC source files or as print images, for ease of documentation. Modification, renaming, and saving of screen images and data field definitions for new or existing programs can take place as required. The user's own source code can be inserted for specialized functions; macro and subroutine libraries are provided and extensible. 6 to 8K are required for storing a program's attributes.

Reports can be directed to terminal, disk, or printer at run time. They include page numbering, dating, formatting, dynamic page breaks, and user-specified record selection.

The record retrieval system, called Micro B+, uses automatic B+tree balancing to fast search without sorts. Indices are maintained without the need for reorganization. Up to fifty keys of 48 characters each are permitted, each with multi-key, partial key and concatenated keys. The number of records allowed is 65,000. 50 fields per record are allowed.

The Programmer's Apprentice runs with a Z80, 8080, or 8085 CPU. I requires 64K RAM, BASCOM Version 5.3, a memory-mapped or cursor addressable terminal, and CP/M-80.

New Publication

Speaking Pascal: A Computer Language Primer
By Kenneth A. Bowen
This is a simple introductory text, explaining the various components of Pascal. Elementary and complex data types, the use of control structures, procedures and functions are discussed. Structured programming techniques are used to create programs. Exercises and illustrations are provided to help the novice along; in addition, an appendix on UCSD is included.

New

Versions

Software authors are urged to send us details on updates they release. This way, users can keep informed of enhancements and bug fixes they may be waiting for. If you don't find an update report here, then the software author hasn't forwarded it to *Lifelines/The Software Magazine*.

BDS C Compiler
Version 1.46

There have been several new sets of features added to BDS C in this version, in three categories: preprocessor enhancement, CP/M-specific compiler performance improvement by selective overwriting of the CCP (Console Command Processor), and new utility programs.

The preprocessor enhancements are as follows:

1- Parameterized #defines are now supported. This allows a macro in the form of a function call to be expanded (before compilation) into an arbitrary string, with the original parameters substituted into the string.
2- One feature of "#define" substitution has been slightly changed: when a symbolic constant appears in the definition of *another* symbolic constant, then the substitution of the first constant does not take place until the substitution of the second does.
3- The "#if <expr>" conditional compilation directive is now supported, but only with a special limited syntax for the expression argument.
4- Nesting of conditional compilation directives is now allowed, and incorrect nesting attempts will now draw an appropriate error instead of doing random things to the source text. Note that each and every #else directive *must* be followed by a matching #endif (unlike C's control structure syntax, in which an if...else chain may be extended as long as desired.)

These enhancements to the compiler and linker affect the use of the compiler, not the C language syntax it accepts:

1- In the past, the compiler and linker have performed a CP/M warm-boot after every compilation had either been completed or aborted due to an error. Now a warm-boot will only take place when the memory occupied by the Console Command Processor (CCP) is actually needed for the task. On certain "fake" CP/M systems (I believe the CROMIX CP/M emulator is one such case), the non-warm-booting return to the CCP does not work correctly, probably because the system does not pass a valid stack pointer to transient commands. Patches to correct this are included in the manual addenda for this version.
2- One feature of BDS C has been that it automatically aborted any pending "SUBMIT" file after compilation, if an error had been detected during the compilation. This feature is no longer automatic, but is optional.
3- The compiler and linker now send a bell character (control-G) to the user console after completing a task in which one or more errors have occurred.
4- Patches are included in the new version to repair the problem with type-ahead during operation of CC1, CC2 or CLINK.

The major new utility program included with this version is CASM.C, an assembly-language-to-CRL conversion preprocessor. CASM takes a specially-formatted assembly language source file with extension and puts out an ".ASM" file which may then be assembled using the standard CP/M assembler (ASM.COM), to eventually produce a CRL-format object file. A separate document detailing the operation of CASM is included with the update.

A new wild-card expansion utility, named WILDEXP.C, allows ambiguous file names to be specified on the command line to C-generated programs; then by a simple function call, the ambiguous references are expanded to include all filenames on the current disk that match the specification. Exceptions may also be specified.

A new utility named NOBOOT.C is also included: when NOBOOT.COM is invoked upon a COM file produced by the C compiler, it will make changes so that the COM file no longer performs a warm-boot after completing execution.

The following bugs have been detected and corrected for BDS C v1.46:

1- CC1 had crashed when an "#include" file was not terminated with a carriage-return/linefeed sequence.
2- CLINK no longer complains about being unable to find "DEFF3.CRL" when there are undefined function references in a linkage.
3- Literal strings having continuation lines might have confused the CC1 preprocessor in some versions, so that a "#defined" symbol name that happened to match a character sequence within the continuation line of the string was incorrectly substituted for by the preprocessor, and such a symbol appearing *after* the end of the string was *not* substituted for.
4- In the DIO package, the variable "c" in the "getchar" function was incorrectly declared as a "char" instead of an "int"; this caused a physical EOF to be returned as the value 255 instead of -1 when the text file was not terminated by a CPMEOF (control-Z) character.
5- Another DIO-related bug: when text containing both carriage-returns and linefeeds was fed to the DIO "putchar" function, an extra linefeed character was appended to each line and resulted in an extra blank line between each line of the output file.
6- CLINK now warns the user when the address of the end of the external data area falls above the effective "top of memory" address to prevent confusion if such a condition is not noticed by the user.
7- The "execl" function had bombed if an attempt was made to pass more

(continued next page)

than six parameters, and it had not detected when the total size of supplied parameters exceeded the amount of space available for that text during the chaining operation (about 83 characters). Now any number of parameters are handled correctly, and a text overflow will cause "execl" to print a special message to that effect and also return a value of ERROR (-1) to the calling routine.

8- The "gets" library function has been modified to use the stack during its BDOS call to get a line of text, and then copy the result into the supplied buffer area. A new alternative to "gets" has been supplied, called "getline", which works just like the "getline" function shown in Kernighan & Ritchie.

More details on all these changes are supplied in the new User's Guide Addenda.

PLAN80
Version 2.2a

This update features a minor change to ensure that the depreciation routines operate with the :FOR statement. Under 2.2 the depreciation calculation routines did not properly ignore rows or columns excluded by a :FOR statement.

T/MAKER II
Version 2.5.3

When a "move column" command was used in the editor in a situation where the new location for the column was to the right of the current location, the old location was deleted before the column was inserted into the new location. This resulted in a shifting of the line which sometimes meant that the column was not inserted in the place you probably wanted it to be. Now the column is copied into its new location before the old is deleted.

Univair Series 9000 Medical and Dental Management Systems
Version 2.0

Office files for the clinic or doctor have been changed so that interest or late charge may be set up by the operator, name and address lines can hold thirty characters, second address lines can be used, and department cost accounting has been added. The cost accounting and name and address features have also been added in doctor and dentist files. The name and address changes are also effected in insurance company files.

Billing messages now can contain up to sixty characters, and an advertising message is permitted on statement print runs. Procedure code files and diagnosis code files now support 8-digit alphanumeric codes with 2-digit modifiers; descriptions may now be thirty characters.

Patient escrow account balances reflect the true account balance of the responsible party. Special funds can be held in escrow by adding the proper amounts to a case in file. There are now nine options for billing and types of accounts, ranging from "no statements to be sent" to "past due account sent out for collection".

Patients now may be located by account number or last name; a SOUNDEX coding system permits the operator to enter a partial spelling of the last name.

Charges now allow for a primary and secondary diagnosis, and an unlimited number of charges are allowed per case. All treatment charges are printed together on insurance forms; department cost accounting has also been added in this category, on cases for more advanced income tracking on general ledger postings.

Pre-planned time payments are permitted now on each case, running from cash in advance through sixty months of equal payments. Automatic interest of late fees can be calculated at the end-of-month billing cycle and added to cases not currently paid up.

Single items may now be edited in the case or charge item; additions can also be made. Adjustments can be made for negative amounts.

Reorganization of indices is only performed at the end of the day, saving time during peak hours.

Inter-office reports have been moved to the master report section for convenience; some changes have been made so that printing reports is easier, and can be halted if necessary. Monthly statements now print a single summary line per case with all total charges added together. More medical insurance forms have been added to those supported, as well as two dental insurance forms.

Advanced linking of case and charge files will provide an increase in available disk space, where there are more than two charges per case.

Bugs

Postmaster
Version 3.5

Under the following conditions the last few labels won't be printed by the PMLABEL program:

- if record extraction is selected
- if labels are being printed more than one across
- if the number of labels that should be printed is not exactly divisible by the number of labels printed across.

For example, if labels are being printed four across and eleven records meet the extraction criteria, PMLABL should print two rows of four label and then one row of three labels. However, the final row of three labels won't be printed.

The remedy is to either print all labels one across; or to rerun PMLABL and print just the last few labels one across.

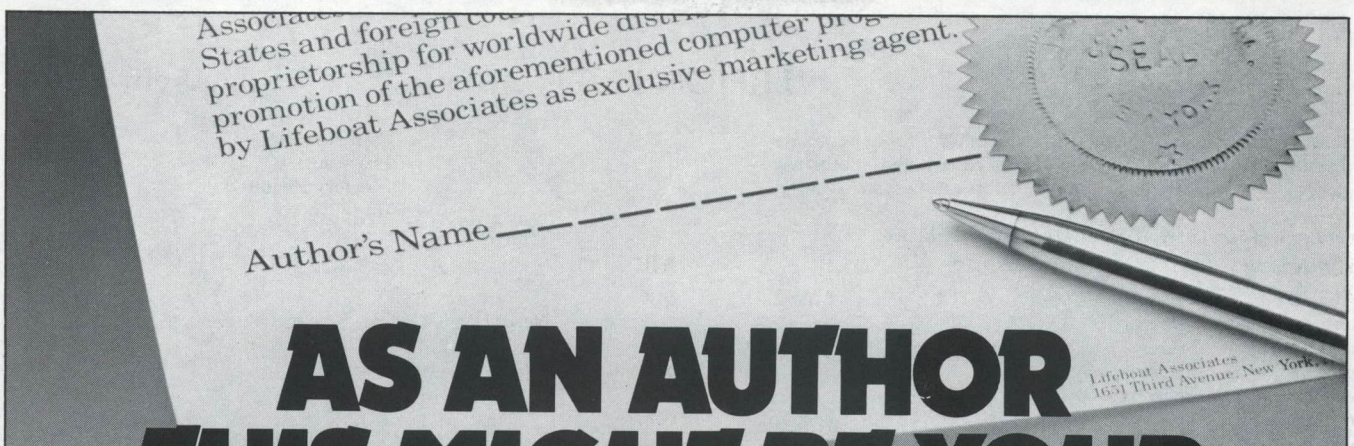
OOPS!

On page 31 of the March issue (Volume II, Number 10), an incorrect address was given for Matthew Von-Maszewski. The right address is Matthew Von-Maszewski, Staley Computer Associates, POB 9158, College Station, TX 77840.

On page 37 of the April issue, in the listing presented by Bob Kowitt, a line is missing:

```
60 GOSUB 560    start byte search
```

In addition, this routine does not work for BASIC-80 with the Apple.



AS AN AUTHOR THIS MIGHT BE YOUR MOST IMPORTANT LINE.

If you've tried to market your own program, you've probably run into a virtual brick wall of problems. Problems that require time, energy, funds, personnel and expertise to solve.

Lifeboat Associates invites you to bring your problems to us. That way you can do what you do best: create quality software. And we can do what we do best: sell it.

As an international publisher of quality computer software with a strong relationship among business, professional, programming and personal computer users, as well as micro- and minicomputer OEM's, Lifeboat Associates has sold and fully supported more software programs by more authors for more machines to more users in more countries than anyone else.

And we do a lot more than sell. Lifeboat also provides:

• Full after sales support • A multitude of media formats • OEM sales • Extensive promotional campaigns through Lifeboat's *Software Desk Reference*™, specially designed OEM private label catalogs, foreign catalogs, brochures, flyers and direct mail • Advertising • Advertising preparation • Marketing services throughout a wide network of affiliates, dealers and distributors • Translation facilities into foreign languages • Seminars • Typesetting services • And lots more

So if you've expended your time and genius in writing a great program, bring it to Lifeboat. We'll expend our time and genius in publishing it.

Write for a copy of the *Lifeboat Author Guide*.

1

Lifeboat Associates

World's foremost software source

1651 Third Avenue, New York, New York 10028

VERSION LIST

April 6, 1982

The listed software is available from the authors, computer stores distributors, and publishers. Except in the cases noted, all software requires CP/M-80, SB-80, or compatible operating systems.

S Standard Version
P Processor
MR Memory Required

New Products and **new versions** are listed in boldface.

Product	S	P	MR	
ACCESS-80	1.0	8080/Z80	54K	
Accounts Payable/Cybernetics				Needs RM/COBOL. Runs w/CP/M-80, OASIS, UNIX
Accounts Payable/MC	1.0	8080/Z80	56K	For CP/M 2.2
Accounts Payable/Structured Sys	1.3B	8080	52K	w/It Works run time pkg.
Accounts Payable/Peachtree	07-13-80		48K	Needs BASIC-80 4.51
Accounting Plus		8080/Z80	64K	
Accounts Receivable/Cybernetics				Needs RM/COBOL. Runs w/CP/M-80, OASIS, UNIX
Accounts Receivable/MC	1.0	8080/Z80	56K	CP/M 2.2
Accounts Receivable/Peachtree	07-13-80	8080	48K	Needs BASIC-80 4.51
Accounts Receivable/Structured Sys	1.4C	8080	56K	w/It Works run time pkg.
Address Management System	1.0	8080		Requires 2 drives
ALGOL 60	4.8C	8080	24K	
ANALYST	2.0	8080	52K	Needs CBASIC2,QSORT/ULTRASORT
APL/V80	3.2	Z80	48K	Needs APL terminal
Apartment Management (Cornwall)	1.0	Z80		Needs CBASIC2
ASM/XITAN	3.11	Z80		
Automated Patient History	1.2	8080	48K	
BASIC Compiler	5.3	8080	48K	
BASIC-80 Interpreter	5.21	8080	40K	w/Vers. 4.51,5.21
BASIC Utility Disk	2.0	8080	48K	
BaZic II	03/03			
Benchmark Word Processor	2.2			Give Name & Model #'s of the video terminal)
Benchmark Mail List	1.1			Give Name & Model #'s of the video terminal)
BOSS Financial Accounting System	1.08	8080	48K	Needs 2/3- drives w/min 200k each, & 132-col. printer
BOSS Demo	1.08	8080	48K	
BSTAM Communication System	4.5	8080	32K	
BDS C Compiler	**1.46**	**8080**	**32K**	**w/'C' book**
Whitesmiths' C Compiler	2.1	8080	60K	
BSTMS	1.2	8080	24K	
BUG / uBUG Debuggers	**3.20**	**Z80**	**24K**	
CBASIC2 Compiler	2.08	8080	32K	w/CRUN(2,204P, & 238)
CBS Applications Builder	1.33	8080	48K	Needs no support language
CIS COBOL Compiler	4.4,1	8080	48K	
CIS COBOL Compact	3.46	8080	32K	
FORMS 1 CIS COBOL Form Generator	1.06	8080		
FORMS 2 CIS COBOL Form Generator	1.1,6a	8080		
Interface for Mits Q70 Printer				CP/M 1.41 or 2.XX
COBOL-80 Compiler	4.6	8080	48K	
COBOL-80 PLUS M/SORT	4.01	8080	48K	
CONDOR II	2.06	8080	48K	
CREAM (Real Estate Acct'ng)	2.3	8080	64K	CBASIC needed
Crosstalk	1.4	Z80		
DATASTAR Information Manager	1.101	8080	48K	
Datebook-II	2.04	8080	48K	Needs 80x24 terminal, N/A for CDOS, CP/M 1.4, MP/M
dBASE-II	2.3B	8080	48K	
dBASE-II Demo	**2.3B**	**8080**	**48K**	
Dental Management System 8000	8.7A	8080	48K	Needs CBASIC
Dental Management System 9000	**2.0**	**8080**	**48K**	**Needs CBASIC**
DESPOOL Print Spooler	2.1A	8080		
DISILOG Z80 Disassembler	4.0	Z80		Zilog mnemonics
DISTEL Z80/8080 Disassembler	4.0	8080/Z80		Intel mnemonics,TDL extensions
Documate/Plus	1.4	8080	36K	
Documate/Plus/Demo	1.5			
EDIT Text Editor	2.06	Z80		
EDIT-80 Text Editor	2.02	8080		
Emulator-86	1.0			An Emulator for CP/M-86
FABS-I	2.7	8080	32K	
FABS II	4.15		8080/Z80	48K
FILETRAN	1.20		32K	1-way TRS-80 Mod I,TRSDOS to Mod I CP/M
FILETRAN	1.4		32K	Needs TRSDOS. 2-way TRS-80 Mod I,TRSDOS & Mod I CP/M
FILETRAN	1.5		32K	1-way TRS-80 Mod II,TRSDOS to Mod II CP/M
Financial Modeling System	2.0		48K	
Floating Point FORTH	2	8080/Z80	28K	
Floating Point FORTH	3	8080/Z80	28K	
FORTRAN-80 Compiler	**3.44**	**8080**	**36K**	
FPL 56K Vers.	2.6	8080	56K	
FPL 48K Vers.	2.6	8080	48K	
General Ledger/Cybernetics				Needs RM/COBOL. Runs w/CP/M-80, OASIS, UNIX
General Ledger/MC	1.0	8080/Z80	56K	Needs CP/M 2.2 or MP/M

VERSION LIST

Product	S	P	MR	
General Ledger/Peachtree	07-13-80	8080	48K	Needs BASIC-80 4.51
General Ledger/Structured Sys	1.4C	8080	52K	w/It Works Package
General Ledger II/CPaids	1.1	8080	48K	Needs BASIC-80 4.51
GLECTOR Accounting System	2.02	8080	56K	Use w/CBASIC2,Selector III
GLECTOR IV Accounting System	1.0	8080		Needs Selector IV
HDBS	1.05A	+	52K	
IBM/CPM	1.1	8080		
Insurance Agency System 9000	1.08	8080		Needs CBASIC
Integrated Acctg Sys/Gen'l Ledger		8080	48K	Needed for 3 pkgs. below
Integrated Acctg Sys/Accts Pyble		8080	48K	
Integrated Acctg Sys/Accts Rcvble		8080	48K	
Integrated Acctg Sys/Payroll		8080	48K	
Interchange		Z80	32K	
Inventory/MicroConsultants	5.3	8080/Z80	56K	Needs CP/M 2.2
Inventory/Peachtree	07-13-80	8080	48K	Needs BASIC-80 4.51
Inventory/Structured Sys	1.0C	8080	52K	w/It Works Package
Job Cost Control System/MC	1.0	8080/Z80	56K	Requires CP/M 2.2
JRT Pascal System	1.4	8080	56K	
LETTERIGHT Text Editor	1.1B	8080	52K	
LINKER		Z80		
MAC	2.0A	8080	20K	
MACRO-80 Macro Assembler Package	3.43	8080/Z80		
MAG/base1 (LMS)	2.0.1	8080	56K	Needs CBASIC, 2.06 or later & 180K/drive
MAG/base2 (IMS)	2.0.1	8080	56K	Needs CBASIC, 2.06 or later & 180K/drive
MAG/base3 (ADS)	2.0.1	8080	56K	Needs CBASIC, 2.06 or later & 180K/drive
Magic Typewriter	3	Z80	48K	
Magic Wand	1.11	8080	32K	
MAG/sam3	4.2	8080	32K	
MAG/sam4	1.1	8080	32K	Needs CBASIC
MAGSORT-C	1.0			For CBASIC
MAGSORT-M	1.0			For MBASIC
MAGSORT-M	1.0			For Compilers — BASCOM, FORTRAN-80, PL/I-80
MAILING ADDRESS Mail List System	07-13-80	8080	48K	
Mail-Merge	3.0	8080		
Master Tax	1.0-80	8080	48K	
Matchmaker		8080	32K	
MDBS	1.05A	+	48K	
MDBS-DRS	1.02	+	52K	
MDBS-QRS	1.0	+	52K	
MDBS-RTL	1.0	+	52K	
MDBS-PKG		+	52K	w/all above MDBS products
Medical Management System 8000	8.7a	8080		Needs CBASIC
Medical Management System 9000	**2.0**	**8080**		**Needs CBASIC**
Microcosm		Z80		CP/M 2.X or MP/M
Microspell	4.3	8080	48K	Needs 150K/drive
Microspell Demo	1.0			For Dealers Only
Microstat	2.08	8080	48K	Needs BASIC-80, 5.03 or later, or CBASIC
Microstat for Apple	2.0			
Mince	2.6	8080	48K	
Mince Demo	2.6	8080	48K	
Mini-Warehouse Mngmt. Sys.	5.5	8080	48K	Needs CBASIC
Money Maestro		8080/Z80	48K	CP/M 1.4 or 2.2
MP/M-I	1.0			
MP/M-II	2.0	8080	48K	Needs MP/M
MSORT	1.01	8080	48K	
Mu LISP-80/Mu STAR Compiler	**2.12**	**8080**		
Mu SIMP / Mu MATH Package	**2.12**	**8080**		**muMATH 80**
NAD Mail List System	3.0D	8080	48K	
Nevada COBOL	2.1	8080	32K	
Order Entry w/Inventory/Cybernetics		Z80		Needs RM/COBOL
Panel	2.2		44K	Also for MP/M
PAS-3 Medical	1.78	8080	56K	Needs 132-col. printer & CBASIC
PAS-3 Dental	1.64	8080	56K	Needs 132-col. printer & CBASIC
PASM Assembler	1.02	Z80		
Pascal/M	4.02	8080	56K	
PASCAL/MT Compiler	3.2	8080	32K	
PASCAL/MT+ w/SPP	5.5	8080	52K	Needs 165K/drive
PASCAL/Z Compiler	4.0	Z80	56K	
Payroll/Cybernetics, Inc.		Z80		Needs RM/COBOL
Payroll/Peachtree	07-13-81	8080	48K	Needs BASIC-80 4.51
Payroll/Structured Sys	1.0E	8080	60K	w/It Works run time pkg.
PEARL SD	3.01	8080	56K	w/CBASIC2,Ultrasort II
PLAN80 Financial Package (Z80/8080)	**2.2A**	**8080**	**56K**	**Z80/8080**
PLAN80 Demo	1.1			
PL/I-80	1.3	8080	48K	
PLINK I Linking Loader	3.28	Z80	24K	

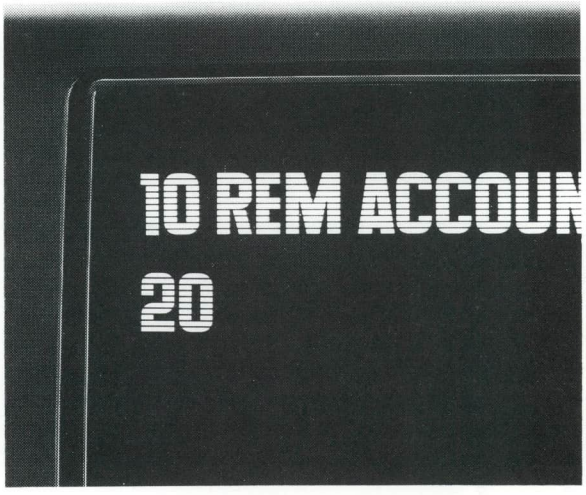
(continued next page)

VERSION LIST

Product	S	P	MR	
PLINK-II Linking Loader	**1.14**	**Z80**	**48K**	
PMATE	3.02	8080	32K	
POSTMASTER Mail List System	3.5	8080	48K	
Professional Time Acctg	3.11a	8080	48K	Needs CBASIC2
Programmer's Apprentice		8080/Z80	56K	Needs BASIC-80
Property Management Program (AMC)	4.2	Z80	48K	Needs CBASIC 2.07+, CP/M-80 2.0+
Property Management System	07-13-80	8080		Needs BASIC-80 4.51
Property Manager	1.0	8080	48K	Needs CBASIC
PSORT	1.3	8080		
QSORT Sort Program	2.0	8080	48K	
Real Estate Acquisition Programs	2.1	8080	56K	Needs CBASIC
Remote	3.01	Z80		
Residential Prop. Mngemt. Sys.	1.0	Z80	48K	
RM/COBOL Compiler				w/Cybernetics CP/M 2, OASIS, UNIX
RAID	5.0.2	8080	28K	
RAID w/FPP	5.0.2	8080	40K	
RECLAIM Disk Verification Program	2.1	8080	16K	
SBASIC	5.4	8080	48K	
Scribble	1.3	8080		
SELECTOR-III-C2 Data Manager	3.24	8080	48K	Needs CBASIC
SELECTOR-IV	2.17	8080	52K	Needs CBASIC
Shortax	1.2	Z80	48K	TRSDOS,MDOS too, needs BASIC-80 5.0
SID Symbolic Debugger	1.4	8080		N/A-Superbr'n
Spellguard	2.0	8080/Z80	32K	Needs Word Processing Program
Standard Tax	1.0	8080	48K	Needs BASIC-80 4.51
STATPAK	1.2	8080		Needs BASIC-80 4.2 or above
STIFF UPPER LISP	2.8	8080	48K	
STRING BIT FORTRAN Routines	1.02	8080		
STRING/80 bit FORTRAN Routines	1.22	8080		
STRING/80 bit Source	1.22	8080		
SUPER SORT I Sort Package	1.5	8080		Max. record=4096 bytes
SELECT		8080/Z80	40K	
T/MAKER II	**2.5.3**	**8080**	**48K**	**Avail. for CDOS**
T/MAKER II DEMO	2.4	8080	48K	
TEX Text Formatter	2.1	8080	36K	
TEXTWRITER-III	3.6	8080	32K	
TINY C Interpreter	800102C	8080		
TINY C-II Compiler	800201	8080		
TRS-80 Customization Disk	1.3C	8080		
ULTRASORT II	4.1C	8080	48K	
Lifeboat Unlock	1.3	8080		Use w/BASIC-80 5.2
VISAM	2.3p	8080	48K	
Wiremaster	3.12	Z80		Needs 180K/drive
Wordindex	3.0	8080	48K	Needs WordStar
Wordmaster	1.07A	8080	40K	
WordStar	3.0	8080	48K	
WordStar w/MailMerge	3.0	8080	48K	
WordStar Customization Notes	3.0	8080		
XASM-05 Cross Assembler	1.05	8080	48K	
XASM-09 Cross Assembler	1.07	8080	48K	
XASM-51 Cross Assembler	1.09	8080	48K	
XASM-F8 Cross Assembler	1.04	8080	48K	
XASM-400 Cross Assembler	1.03	8080	48K	
XASM-18 Cross Assembler	1.41	8080		
XASM-48 Cross Assembler	1.62	8080		
XASM-65 Cross Assembler	1.97	8080		
XASM-68 Cross Assembler	2.00	8080		
XYBASIC Extended Interpreter	2.11	8080		
XYBASIC Extended Disk Interpreter	2.11	8080		With EDIT features
XYBASIC Extended Compiler	2.0	8080		Requires the XYBASIC w/EDIT features to create SOURCE
XYBASIC Extended Romable	2.1	8080		
XYBASIC Integer Interpreter	1.7	8080		
XYBASIC Integer Compiler	2.0	8080		
XYBASIC Integer Romable	1.7	8080		
ZAP-80	1.4	8080		Needs 50K/drive
Z80 Development Package	3.5	Z80		N/A-Magnolia,Superbr'n,mod.CP/M
ZDM/ZDMZ Debugger	1.2/2.0	Z80		For N'Star,Apple,IBM 8"
ZDT Z80 Debugger	1.41	1.41	Z80	N/A-Superbr'n,mod.CP/M
ZSID Z80 Debugger	1.4A	Z80		N/A-Superbr'n,mod.CP/M

+These products are available in Z80 or 8080, in the following host language: BASCOM, COBOL-80, FORTRAN-80, PASCAL/M, PASCAL/Z, CIS-COBOL, CBASIC, PL/I-80, BASIC-80 4.51, and BASIC-80 5.xx.

BOY, IS THIS COSTING YOU.



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

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

Also available from Lifeboat Associates.

1651 Third Avenue / New York, N.Y. 10028

Second Class Postage Paid
At New York, N.Y.