LIFELINES™
The Software Magazine™

$2.50 March 1982 Volume II, No. 10 (ISSN 0279-2575, USPS 597-830)

WHY PLAN 80?

PLAN80™ is a new system that takes the big business, big computer approach to computer modeling and adapts it to smaller computers, which are inherently more friendly and responsive.

If you are not already familiar with the world of financial modeling you will soon wonder how you managed without a system like PLAN80. If you are familiar with the art you will find it incredible that a microcomputer can do so much of what has previously been the domain of million dollar machines.

PLAN80 WILL DO 99% OF THE JOBS DONE BY COMPUTER MODELING SYSTEMS COSTING $50,000

Check your interests:

- ☐ Profit Planning
- ☐ Cash Management
- ☐ Acquisition Analysis
- ☐ Market Simulation
- ☐ Resource Allocation
- ☐ Lease vs. Purchase Analysis
- ☐ Purchase Price Trends
- ☐ Balance Sheet Projection
- ☐ Cost Center Budgeting
- ☐ Productivity Trend Analysis
- ☐ Sales Projection and Analysis
- ☐ Marketing Strategy Development
- ☐ Capital Project Evaluation
- ☐ Headcount Analysis and Control
- ☐ Cost and Variance Analysis
- ☐ R&D Project Evaluation
- ☐ Energy Accounting
- ☐ Cost Estimating
- ☐ Consolidations
- ☐ Tax Planning

☐ Please send more information about PLAN80

☐ Please send a Software Desk Reference™

Dealer, Distributor, and OEM inquiries invited

Please note: All Lifeboat Associates microcomputer software requires SB-80™ or other CP/M-80® compatible operating system.

And soon versions will be available for IBM Personal Computer DOS (SB-86,™ MS-DOS™), and other systems as well.

NAME ______________________

TITLE ______________________

COMPANY ______________________

STREET ______________________

CITY ________ STATE ____ ZIP ____

For More Information, contact

LIFEBOAT ASSOCIATES
1651 Third Avenue
New York, New York 10028
Tel: (212) 860-0300
TWX: 710-581-2524 (LBSOFT NYK)
Telex: 640693 (LBSOFT NYK)

LIFELINES
The Software Magazine

March 1982

Volume II, No. 10

Editor-in-Chief: Edward H. Currie
Managing Editor: Jane Mellin
Administrative Assistant: Patricia Matthews
Production Assistant: K. Gartner
Typographer: Harold Black

Cover by K. Gartner

Contents

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.

Editorial Comments

There Are More Things in Heaven and Earth Horatio ...

The trade publications are showing a gratifying interest in our pursuit of standards. As a result of *Lifelines'* call for standards many of the leaders in our industry are now actively engaged in discussions regarding standards for the micro industry; and operating systems are at the top of their lists.

A number of you have requested some discussion of the language Ada, which has recently been adopted by the Department of Defense as a standard. This interesting language was designed with three primary goals in mind: efficiency, reliability and ease of maintenance, along with recognition that programs are prepared by programmers. Readability was emphasized over ease of writing. For example, program variables must be explicitly declared and their type specified with a specific prohibition against automatic type conversion. This guarantees that objects are used only as intended. English-like constructs are used throughout to facilitate readability. Particular emphasis was placed on keeping the language as small as possible.

Ada has many of the constructs found in languages such as Euclid, Lis, Mesa, Modula and Sue; these are Pascal derivatives. Ada is truly the product of a language design engineering project and is thought by many to represent the state of the art in languages.

The design of Ada was also influenced by ALGOL 68 and Simula, as well as Alphard and Clu – but not to the extent of the Pascal derivatives.

An Ada program consists of a sequence of higher level program units, each of which may be separately compiled. Such program units may in fact be subprograms defining executable algorithms, so-called package modules defining collections of objects, or task modules defining concurrent computations.

CP/M-80 compatible versions are already emerging and should prove of great interest to Pascal lovers. Take a look at the recent series of articles in Microsystems on Little-Ada by Ralph Kenyon. He has provided a run time interpreter and source code and will in the final installment present the compiler and object code.

Not everyone is enthusiastic about this new entry, however. In a recent issue of an English technical publication an author suggested that there was genuine cause for concern about the implementation of Ada by the Department of Defense. The author's concern is that this new language is being used to program nuclear warhead missiles. He suggests that a language as new and as complex as Ada is potentially quite dangerous and not really the sort of software to be used for programming nuclear weapons. This could be the source of the final indignation by mankind resulting from the manifestation of the "ultimate bug".

The field of ham radio has undergone drastic changes in recent years with the advent of the microcomputer; unfortunately, though some had predicted rapid expansion of this amateur hobby it appears to be in fact contracting at an alarming rate. Apparently many hams have abandoned their former hobby and moved into the limitless realm of micros. The FCC has, as usual, been slow to react to this and has only recently done anything which would attract computerists into this area. Unfortunately, their efforts may be too little and too late. ASCII transmission at three hundred baud seems a pathetic concession, particularly when the Morse code requirements remain. We can only hope that the FCC will move rapidly in removing these archaic barriers to those who would breathe new life into this historically important avocation. If they will lift the code requirements and permit 4800 or 9600 baud transmission there is a good chance that a whole new area of amateur radio could emerge analogous to that of its telecommunications counterpart, i.e. CBBSs. Those concerned that the airwaves not be turned into another CB-land could be reassured by perhaps additional written examination requirements to filter out those who are more amused by abuse than by responsible exploitation. Imagine being able to tune into the broadcasts of the entire CPMUG library at 9600 baud. Programs could be prepared for your computer that would wait for the appropriate codes to be received and then begin writing the volume(s) to disk. Satellite communications would also be of interest to those around the globe looking for low cost, high speed program transmission.

Those of you concerned with this problem should make your feelings known to the FCC.

It's interesting to contemplate what area of hobbyist activity will follow microcomputers. It's rather difficult to imagine what technology could surpass the microcomputer. If you have any ideas in this area please let me know and we'll review them in a future editorial.

The interest in SB-86 (a.k.a. MSDOS) is continuing to grow rapidly and we are gratified by the response of the readership, which has been very supportive of editorials discussing the various aspects of this most important entry into the area of microcomputer operating systems. There is now a CP/Emulator which is a program designed to run under MSDOS to permit CP/M-86 programs to not only run under MSDOS but to run faster ... and the beat goes on ...

The CBBSs are now offering their standard fare in a "crunched format which saves you about 20-40 percent in ASCII file size, an important consideration if it's your nickel that the transmission is being played on. These guys seem to have no end of innovation to bring to the game. So the next time that you dial in, watch for a couple of files called SQ.COM and USQ.COM. They will save you a bundle. The basic technique is to crunch all those spaces which seem to make up a surprisingly significant portion of ASCII files. Once you have received the file you have only to use USQ.COM and presto, it's unscrambled!

It will be interesting how long it will take for the first 8086/8088 based CBBS to appear (running SB-86 of course). With the ease with which all the communications programs can be translated and reassembled it won't be long.

A number of you have asked about STOIC, which was mentioned in a previous editorial. Wink Saville has a later version which he might be persuaded to make available if enough of you are interested.

Keep sending those cards and letters, folks ...

Edward H. Currie

The Pipeline

Carl Warren

Resources aplenty and a low cost network

One of the most difficult tasks I have as a trade journalist is keeping track of the mounds of information I need to perform my job. Some of this information comes from other magazines, especially when I'm researching a story on a product segment, or developing a reference list.

In the past I, like you, have had to dig through my ever-growing mound of magazines to find out which articles were printed where and how they went about presenting the material. I have even gone to all the trouble of promising my secretary great wealth and fame if she could bring some semblance of order to this mountain of mayhem.

As could be expected from any intrepid secretary she was more than up to the task. Her first order of business was putting everything in order, a task that I have found to be perplexing. Next, to assist me in finding where which article appeared, she sprung for a $14.95 copy of "The Index" by W.H. Wallace, indexor. This book is published by Missouri Indexing Inc, Box 301, St. Ann, MO 63074 (314) 997-6470 and contains over 30,000 entries encompassing 6 years of articles and editorials which have appeared in more than 800 issues of personal computer magazines - including most of my own writings.

The book has already paid for itself. Unfortunately, I now have to come up with the fame and fortune that I promised.

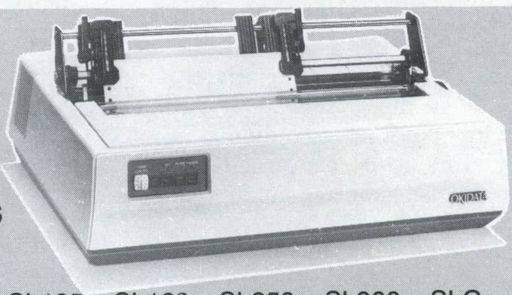
Since I'm embarking on a unique set of articles for this column, I used The Index to find out if anyone had ever considered publishing a guide to I/O devices. What I did find were a number of articles discussing some devices, but nothing giving an overall look. But more on this later.

Since I had one of my major research problems partially solved, with the aid of secretary Brenda and The Index, I now was looking for something similar for hardware. As luck would have it (and I can assure you editors rely on luck) I was talking to Bill Godbout and mentioned that it would be nice if someone had put together a compendium of hardware manuals on various equipment.

It turned out Bill had done just that with his products, spanning from 1975 to September of 1981. The books called Product User Manuals Vol 1 and 2 cover the S-100 products produced by Godbout Electronics over this time frame. Vol 1, priced at $20,

(continued on next page)

OKIDATA •APART FROM THE REST!



A FULL LINE OF 100% DUTY CYCLE PRINTERS

MODEL	ML80	ML82A	ML83A	ML84	SL125	SL160	SL250	SL300	SLG
COLUMNS:	80	80	132	132	132	132	132	132	132
THROUGHPUT: (lpm)									
20 Char/line	86	232	232	266					
40 Char/line	51	138	138	184					
80 Char/line	28	76	76	114					
132 Char/line			47	74	125	160	250	300	400
DUTY CYCLE (%)	100	100	100	100	100	100	100	100	100
HEAD WARRANTY	— 200 million characters —				— 500 million characters —				
GRAPHICS:	✓	✓	✓	✓	✓	—	✓	—	✓
RS 232:	Opt.	Std.	Std.	Opt.	Opt.	Opt.	Opt.	Opt.	Opt.
FRICTION FEED:	✓	✓	✓	✓	—	—	—	—	—
TRACTOR FEED:	Opt.	Opt.	✓	✓	✓	✓	✓	✓	✓
PIN FEED:	✓	✓	—	—	—	—	—	—	—

DISTRIBUTED by: **GRAYDON-SHERMAN, INC.**

(212) 289-3199 (201) 467-1401 TWX #710-983-4375 (GRAYDON MAWD)

covers the famous Econoram series to the CPU-Z and includes full user information with schematics, timing diagrams, and, where necessary, software. Vol 2, priced at $25, goes the next step and even covers the dual 8085/88 processor board that has found so much favor among software developers. With the series beginning next month I'll be drawing heavily from this two volume set; so you might give Bill a call at (415) 562-0636.

A very low cost network server

You have probably noticed that the watch-word for the year is networking. The concept isn't new, but many of the implementation schemes are. Already the much-touted Ethernet has been overshadowed by less expensive, and in many cases more capable designs.

Interestingly, most of the schemes developed so far require a host adaptor, a transmission box of some sort, more expense, and very specialized software. The data rates run from a low of 56K bytes/sec to 10M bytes/sec in very expensive Ethernet implementations.

I've found, however, that very few applications require transmission rates above 19.2K baud, and that the incidence of collision is relatively low. Moreover, most users aren't planning to wire up a 55 story building, nor do they want to be concerned over the characteristics of transmission lines.

What most potential network users want above anything else is low cost. And they are willing to make some tradeoffs to get the desired price/performance.

Some of the tradeoffs that have to be made are in the area of speed. The old axiom of "as the speed goes up so does the price" holds very true in networking systems. In addition, response time must be considered. What has to be determined is: do you really need instant access all the time? More than likely in an office with a dozen or so micros lashed together the answer is no.

Apparently, Radio Shack has come to the same conclusions as I have. And I congratulate them on their far-sightedness. Radio Shack is offering the Network III for $599.

The Network III is a software controlled switch for multiplex 16-RS 232C serial channels operating at 19.2K baud. This means that for about $37.43 per connection, you can combine 16 micros, terminals, and printers for a very low-cost networking scheme.

As sold, the Network III comes with software designed to work with TRS-80 Model IIIs. But you aren't restricted to the Radio Shack equipment. You can use any device you like and any processor; the only requirement is an RS-232 Port and the correct software.

The software used is designed to poll each device in turn to see if it is vying for the network or can accept a message, or is planning to send a message to another unit on the system, like the master who also serves as the file server, or a printer.

Collisions are avoided in the network by pulling Data Terminal Ready (DTR) low, on all but the active channels, and coupling the two talking and listening devices together. When transmission is finished, the master then brings DTR high on all the devices, enabling them to talk to the network.

Since a polling scheme is used, priority is based on which device is connected to what channel, and you can develop software that interleaves the polled channels so that devices which will have the greatest activity will be polled first and always have priority over the others, even instigating a pseudo interrupt to cause a network grant.

The one unfortunate thing about the Radio Shack implementation is that it requires one device to handle all the work of the system. But not to fear, a more stand-alone device is on the way, from an Orange County, CA-based design firm.

Reportedly, this device will use a built-in microprocessor, have RAM buffering, and will handle all polling and message switching tasks. From what we understand, the device will accept control codes to perform specific tasks, much like several intelligent modems do now.

The device is under development now with an expected introduction sometime in late summer, for a price tag below $400. Yes we know who the manufacturer is, and for now they have asked that that piece of information not be disclosed. And no, it isn't a Japanese firm.

The software piracy business

Like networking, software piracy has been getting a fair share of attention in the press lately. And has generated many possible methods of locking software up from unwanted stealing.

One of the best methods to prevent stealing was suggested by San Francisco based consultant Jim Edlin. He believes, and I agree, that possibly software is overpriced in the first place. Consequently, it becomes easy for a group to pool their money, buy one copy and spread it around. Their conscience is clear after all. They all contributed - didn't they?

What they did do is lower the revenues of some coder who spent hundreds of hours developing the package, only to lose in the long run. What if the package was priced at $100 instead of $400? Possibly that same group would pool their money to buy many copies rather than just one - and of course everyone would be happy.

Two methods that I've heard bantered around a great deal are: first, remove all copyrights, lower the price and encourage copying; second, form a trade group made up of software publishers to force computer makers to provide a special hardware enhancement that provides a combination hardware/software lock. This way a piece of software must be specifically coded to that unique device to operate.

Both are intriguing ideas. Both are workable. Both would change the economic picture for software designers drastically. Personally I think the first is the most acceptable but is the most unworkable in realistic terms. The second is also acceptable but would add another unwanted price onto micros when the name of the game is lower price tags. The answer for now is that no one has one.

Something I ran across

Lately my friend Pat McMullen and I

(continued on page 48)

Full Screen Program Editors: WordMaster

Ward Christensen

WordMaster was my first full screen editor. I have been using it for over three years. I therefore tend to use it as the ruler by which I measure other editors. I will try to forget my slightly biased attachment to it. I'll frequently refer to it below as "WM".

This evaluation is based on WM version 1.07, to which I recently upgraded, from my three-and-a-half year old version 1.02. The $25 upgrade included a new disk and a replacement manual.

Evaluation

DOCUMENTATION: Includes a 56 page documentation and installation manual, and a 28 page operator's guide.

It introduces the three operating modes: (1) video, (2) command, and (3) insert. Editing concepts are covered adequately for the novice. I found the one-page summary of commands, and a similar page of video mode control keys to be a handy reference.

WM is unique in having an on-screen HELP facility, activated by pressing control-J. It offers all the "memory jogger" help you would need to avoid looking back at the documentation.

The operator's manual features step-by-step learning, with exercises. The manual includes a very helpful "visual picture" keyboard layout, labeled with the video mode key functions.

SPEED: WordMaster is very fast. In command mode, it beats all the rest. Video mode is quite fast, but doesn't support terminal hardware line insert and delete. Part of the speed is due to the simple, straightforward display mode: no status line, just text. The bottom two lines are unused, presumably so that when you escape to command mode, the screen doesn't scroll.

ERGONOMICS: In several years of using WordMaster, I have certainly found nothing annoying about it.

It is comfortable to use, and doesn't make your fingers perform unnatural acts. The presence of the control-J help key aids both the learner, and the user who doesn't use it frequently enough to remember all the commands.

MicroPro is to be congratulated for choosing to make the command mode an extension of that of ED. Only the following minor changes were made: (1) control-N is used to represent "carriage return linefeed" (CR/LF) in find and substitute commands, where ED used control-L; (2) ESC may be used to separate strings, where ED used control-Z; (WM still allows control-Z for compatibility); (3) The rather powerful, but seldom used ED "J" (Juxtaposition) command was not implemented.

CONFIGURABILITY: WordMaster comes with configuration files for the ADM3A (and its look-alikes, the Soroc IQ/120 and the IMSAI VIO), for the Hazeltine 1500, the SOL/VDM, and the Beehive 150.

WordMaster, as shipped, supports the ADM. Modifying it for one of the other supported terminals requires doing an assembly, and using DDT or SID to install the patches.

Configuration for other terminals is accomplished by editing one of the supplied 8080 ASM files, and using ASM and DDT (or SID) to make the patches.

The keyboard for WordMaster is *not* configurable. You have to be satisfied with what they have given you.

Older versions (such as my old 1.02) used the "right hand" for cursor movement — control H, J, K, and L being used for cursor back, down, up, and right. Newer versions use "left hand" cursor, keys, to be compatible with WordStar.

Although the WM is not keyboard configurable, I have figured out how to do it, and for a "hacker", it is not very difficult. I'll tell how, near the end of this review.

EASY TO LEARN: The command mode is very straightforward. If you have worked with ED, it will be a snap.

Video mode is quite easy to learn. The "keyboard picture" in the manual lets you visualize how it has been organized: the left hand controls all the file, screen, and cursor movement, while the right hand does the more "obscure" functions — line delete, insert toggle, repeat, show menu, etc. An exception is control-T, a left-hand key for deleting the word to the right of the cursor.

MicroPro was very thoughtful in allowing two of the most common cursor movement keys (cursor left and cursor right) to be done by either hand. Control-S or control-H may be used to move the cursor left, and control-D or control-L to move the cursor right.

Objective Criteria
Video Related Criteria

FULL SCREEN: There are keys to move the cursor one line up or down, one character left or right, one word left or right.

Another key moves the cursor to the top of the page, or if already there, to the bottom. Yet another key moves to the front of the current line, or if already there, to the end. I like this "economizing" as it doesn't take four keys to perform the functions two can do.

The repeat key (see below) may be used to extend these functions.

SCROLLING: There are keys to scroll one line or the entire screen, up or down. The repeat key (see below) may be used to extend these functions. To go to the top or bottom of the file

(continued next page)

requires hitting "a lot" of screen up or down keys, or exiting to WM command mode, and typing a command. For example, to go to the top of the file, hit "ESC b v return". The ESC escapes from video mode to command mode. The "b" goes to the beginning of the file, and the "v" puts you back in video mode, and return ends the command.

WordMaster supports full bi-directional file scrolling. You can proceed from the top to the bottom and back, of the the largest file, without a worry (as long as you have enough disk space). If you just go forward from the top to the bottom of a big file, you will only require twice the file size on disk. Full scrolling requires the "reverse scrolling" file to be created, which takes one file's worth of space, minus the memory buffer size.

If you are tight on disk space, it is not possible to confine yourself to not using the backwards-scrolling file, unless you simply avoid scrolling back. You may scroll back a little, but can "stop" when you hear the disk access, and thus avoid using very much disk space. The "H" command may be used to position to the front of the file, while using a minimum of disk space. The backup file is, however, overwritten.

INSERT: WM uses a key, control-O, to "toggle" into or back out of insert mode. In insert mode, all characters insert, including return.

OVERTYPE: In non-insert mode, all character keys you press, including tab, "overtype" what is under the cursor. Only carriage return doesn't overtype, instead taking you to the front of the next line. You can't actually overtype a carriage return, but may use control-N to force a new line without having to go into insert mode.

UNDO-KEY. WordMaster simply doesn't have one. If you accidentally press control-K which erases to the end of the current line, your only option is re-key in what was lost, or to quit the edit.

REPEAT: In video mode, a very handy "repeat" key applies. Pressing control-@ causes WM to treat the next key-press as if you had typed it four times. This also applies to control-@ itself, allowing four, sixteen, or sixty-four repeats. It works with *all* keys: cursor movement, scrolling, deleting, etc. It also works with character keys, so to generate an "arrow" like "→" requires hitting control-@, "=", and ">". To put a line of sixty-four dashes in a file requires only hitting control-@ three times, then pressing "-", then return.
(Ed: This doesn't work on all terminals.)

TEXT EDITING ABILITIES: WM has very few special abilities oriented toward text. It does have word tab forward or backward, and word delete forward or backward.

No "word wrap" or "fill" mode is supported. However, like any editor which has command characters, you can key in some primitive text formatting commands.

An example: suppose you have keyed in some text, edited it, and now would like to try to make the right margin a bit more even.

A command string of:

```
ms↑N$ $0l72c-s $↑N$vz
```

will, one by one, combine two lines, then cut them off at the nearest space preceding column 72.

This bears some explanation. It's not the kind of macro you would likely think of until you had spent some time with WM, but is an example of the flexibility offered by a good command language. Because spaces are difficult to show in print, I'll put "—" in the following example, to mean a space. In actually executing the macro, you'd have to use a real space, not "—".

m	Macro: over and over
s↑N$	Search for CR/LF
—$	Change it to a space
0L	Move to front of line
72c	Move ahead 72 chars.
-s—$	Reverse search space
↑N$	Change it to CR/LF
v	Go back to video mode
z	"Sleep" when exiting video mode, for 1 sec.

That is all pretty straightforward, except perhaps for the "vz". That is a trick I learned. When WM encounters a "v" in a command string, it switches from command mode to video mode, and temporarily suspends the command execution. When you eventually press ESC to return to command mode, the command continues.

At some point, you would like to stop the command from executing. Pressing control-C will stop it, but at times you might not be quick enough, and the macro will appear to "run away".

The "z" command tells WM to "sleep" for a second. This second gives you ample time to press control-C, which WM will check for upon "waking up" from the sleep. This provides a very clean exit from a macro, at the cost of taking an extra second for each line processed.

Alternatively, you could put a specific number corresponding to the number of lines in a paragraph, in front of the "m". However, there are tradeoffs: while this will now quickly process the paragraph, you will have to type the entire command in again for the next paragraph.

Command Related Criteria

MOVE: You may move to the top or bottom of the file (via "B" or "-B"), ahead or back by character (via "C" or "-C"), or line (via "L" or "-L"). In addition, "0L" moves to the front of the current line. A positive or negative number may precede the command, or a "#" to indicate 65535.

Giving just a positive or negative number, with no command letter, moves that many lines, and types the line moved to.

WM may also move and display a full screen at a time in command mode, using the "P" (page) command. "P" moves down one full screen, and types the next screen. "0P" displays the current screen and doesn't move. "-P" moves back a page, and displays it.

DELETE: WM may delete characters (via "D") or lines (via "K" — meaning kill). 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 all the remaining lines in the document.

INSERT: Arbitrary character strings may be inserted. The "I" command inserts the characters following it, up to a terminating ESC or control-Z.

Also, an arbitrary control character may be inserted into a file, by typing "n!" where "n" is a decimal number.

TYPE: The "T" command types from the cursor to the end of the line. "0T" types from the beginning of the line to the cursor. Thus, "0TT" types the entire cursor line, without moving the cursor. An alternative, "0LT" moves to the front of the line, and types it.

A number preceding the "T" indicates how many lines to type. The file positioning is not changed, i.e. "9T" types 9 lines, but leaves you positioned to the first one shown.

"T" supplies one of the ways of determining how many lines might be needed for a K (line kill) or W (file write) command.

For example, suppose you want to write some exact part of the file to another file. Since the "W" command requires a line count, you might "150T" and see how far that goes. If you see it went 4 lines too far, you can then "146Wfilename" to write out the selected block.

As was mentioned above under "MOVE", the "P" command may be used to type one page. "0P" types one screen from the cursor. "P" moves down one screen and types the next.

The contents of the scratchpad, mentioned below under MOVE and COPY, may be typed, using a "QT" command.

FIND: WM can find a string "short" or "long". Use a "short" find (command letter "F") if you know that what you are looking for is not too far away. It will look at least 2,000 characters ahead, but won't pursue reading in more of the file if that is not enough.

A "long" find ("N", for "next") will continue to search until it finds the character string, or reaches the bottom of the file. You can interrupt out of the search if you feel it is going too far.

Either find may be done forward or backward. An optional number before the "F" or "N" specifies finding the "nth" occurrence.

Additionally, certain special characters are allowed, to "match any character", or "match any character except a specific one", or to match a "spacing" character (space, tab, new line).

CHANGE: The change command is similar to the FIND command, in that it has both short and long variations, and may be done either ahead or back. "S" (substitute) is the "short" change command, and "R" (replace) is the "long" change command.

The format is to type the command letter (S or R), then the string to search for, then a delimiter (either ESC, or control-Z), then the text to be substituted, again followed by a delimiter. (Typing "0TT" following that, will show the line after the change, without having to go into video mode)

MOVE and COPY: WordMaster has a second buffer, which it calls a scratchpad. It is accessed via "Q" commands.

There are commands to put "n" lines in: "nQP", to get the scratchpad back "n" times: "nQG", and to append "n" lines to the scratchpad: "n/QP".

This allows moving text by putting it to the scratchpad (nQP), then moving somewhere else and getting it (QG).

COPY is performed simply by putting lines in the scratchpad, then getting them back into their original position, before positioning to where the lines are to be copied. No facility exists for tagging blocks — you have to use a line count.

The space for the scratchpad is managed automatically, and does not take away from the editing space if it is not used.

Large copies may be done using a scratch disk file. See READ and WRITE below.

COMMAND STRINGS: The above command may be combined into flexible "little programs".

You can repeat an entire string by starting it with "M" for "macro", or with a number, then "M", to repeat that many times.

An example: To scan the file, changing every "foo" to "zot", and then type the line containing the change, type:

```
msfoo$zot$0tt
```

where "$" means ESC. Taking this apart, it means:

m	"Macro": over and over
sfoo$	search for foo
zot$	replace with zot
0t	type line to the left of the cursor
t	type line to the right of the cursor

Additionally, nesting may be performed, using "angle brackets" (greater than and less than signs). For example, to insert 12 rows of 60 dashes in a file, you type:

```
12<60<i-$>i↑N$>
```

↑N means a carriage return, linefeed combination. Taking this one apart, it means:

12<	repeat 12 times
60<	repeat 60 times
i-$	insert "-"
>	end of 60 repeat string
i↑N$	insert final CR/LF
>	end of 12 repeat string

MULTIPLE EDITS: WordMaster is not explicitly capable of editing multiple files in one edit session. However, if the files to edit are small enough to fit in memory, and you don't need backups, there is a way. I use this technique to maintain the message files on CBBS.

First, execute WM with a "dummy" filename:

```
WM FOO
```

WM will say it is a new file, and put you in video mode. Just ESC to command mode, and "Yank" (discussed under READ below) in what you want to edit:

```
YMESSAGE.X64
```

Then, do your normal full screen and command editing, but when you are done, don't "E" (end) but rather just write the file back:

```
B#WMESSAGE.X64
```

"B" gets you to the top of the file, and #W writes the rest of the file (i.e. all of it) to "MESSAGE.X64". Crude, but it works.

(continued next page)

File Related Criteria

BACKUP: WordMaster acts like ED, in that it creates a temporary file for output. When you end, using the "E" command, WM renames your original file to ".BAK", and renames the new file to the original file's name.

WM may be invoked, specifying a second drive on which to place the new output file. This may be done when both the file and the backup won't fit on one disk.

SAVE: There is no explicit SAVE command in WM, but the "H" command, to move to the "head" of the file, writes out what you have done to the temporary file, erases the original input file, renames the output file to the input file name, and creates a new temporary file. (Takes longer to tell about than it takes to execute!)

This process is identical to, but much faster than, ending edit and then going back into WM.

QUIT: WM has two ways to quit: The "O" (original) command keeps you in WM, but tosses any (presumably wrong) changes you have made. The "Q" (quit) command erases the temporary output file, and returns to CP/M, having left the input file unchanged (unless you had executed "H" commands — see SAVE above).

READ: ED has the ability to bring in only .LIB files. WordMaster extends the file read ability to any file on any disk. A "Y" (yank) command specifies the name of the file to be read. If the file type is not specified, ".LIB" is the default. There is no provision for including only part of the file.

WRITE: WordMaster can write an arbitrary number of lines to any file. When you wish to write the entire remainder of the file to some other file, "#Wfilename" may be used. However, if the number of lines is quite large, but doesn't go to the end of the file, you will have to go through some pains to determine the number of lines, as I mentioned under "TYPE" above.

DIRECTORY: WordMaster has no abilities to access the directory, change disks, or erase files.

Statistics

WordMaster retails for $150. It takes only 10K of memory, leaving quite a lot free for your program. It is available in standard, as well as 4200H offset CP/M versions for TRS-80 and H-89.

Room for Improvement

WordMaster has no serious faults. Although still "leading the pack" in raw performance, it is getting a bit "dated", lacking some of the more "modern" characteristics: (1) terminal customization without having to use ASM and DDT; (2) a wider selection of terminals; (3) customizable keyboard; (4) "UNDO" key to allow you to correct minor boo-boos; (5) block marking to facilitate move, copy, delete and file writing; (6) support for terminal line insert and delete; (7) support for (perhaps somewhat archaic) memory mapped terminals; (8) support for some terminals multi-keystroke function keys, i.e. keys which send in "ESC-such-and-such". (NOTE: the H-89 version does support some of its function keys quite nicely).

A comment on the lack of line insert/delete hardware support: Without it, WM must re-draw the entire screen when you scroll backward. However, it implements a "preempt scheme", which means that typing a command which would further change the display, preempts the previous command. Thus, pressing "scroll back" several times, or using a terminal repeat key, interrupts scrolling. When you finally release the key, the screen makes its final update.

There is no special support for memory mapped terminals, and therefore not the "blinding speed" which PMATE, for example, exhibits on memory mapped terminals. It is therefore necessary to support your memory mapped board as some kind of terminal, preferably an ADM-3, which will save you having to customize WordMaster.

Recommendations

WordMaster is still a good compromise between speed, size, and function. It is advantageous being only 10K, as you can edit quite a large file with no extra disk paging.

If I were to recommend changes, I'd suggest that MicroPro add: (1) marked blocks for move, copy, delete, and write; and (2) support for hardware line insert and delete.

A Hacker's Guide to Configuring the Keyboard

If it is important to you to customize the key layout, WordMaster is not for you, unless you are a "hacker" — someone willing to "dink around" with ASM and DDT.

Although not documented, the structure of WordMaster lends itself to "hack" keyboard customization.

There are two areas in the .COM file that have to be changed.

The first is a "priority" table, and gives some of the characteristics of various keys, such as whether the action it produces is preemptable.

The second is a table of subroutine addresses, which handle each control key.

Both tables are organized in ASCII order, with control-@ first, then control-A, control-B, and so on, through control-Z, and the several control keys after control-Z.

To customize your keyboard, you have to find the priority table entry, and subroutine address, for a particular key, then rearrange the table entries to the new location corresponding to the control key you have assigned for the function.

The first table contains one-byte entries. They have values of 0, 1, 5, 8, 20H, 30H, 40H, or 80H. For example, 80H means the action produced is preemptable. The second table contains subroutine addresses, i.e. are two-byte entries.

Here are the addresses of the tables, for the two versions of WM which I have purchased:

Version:	1.02	1.07
Priority:	102DH	10D6H
Addresses:	111CH	11C5H

A Minor Bug

I must say, that my version 1.02 of

WordMaster was completely bug free.

The version 1.07, which I have not yet had time to use very much, has a very minor bug, specifically in the priority table mentioned above.

MicroPro missed setting the preempt bit, for the control-W scroll up key, probably when they changed WM from the "right hand" version, to the "left hand" version.

Thus, in version 1.02, if you pressed control-E (scroll up) several times, the display would quickly catch up to you. In version 1.07, the scroll-up key, control-W, scrolls over and over and over, once for each key press, rather than preempting. It eventually catches up to you, so there is no "integrity" problem.

I have just informed MicroPro of this bug, and presented a fix: simply patch address 10EDH from 00 to 80H.

"Beneath the Covers"

Are you curious what makes WM tick? ...or even ED?

This data is not relevant to the review, but gives a bit of insight into the products.

Important to their speed is the way in which the data is kept in memory.

If you or I were to sit down and invent an editor, we might be inclined to just put all the data "down low" in memory, then, when doing something with the file, move the data as necessary.

For example, to insert a character, just "slide" memory down to make room for it.

Digital Research, with ED, and MicroPro, with WordMaster, were much smarter than that.

They consider the cursor location to be very important. After all, if you are going to go into insert mode, the cursor position is where the new data will be placed. Similarly, a file read, or in WM, a scratchpad GET, will go in at the cursor position.

So, they treat the file like a file of 3x5 cards, opening it up at the place of insert. The part of the file that is "behind" the cursor, is pushed against the front of memory, right behind the editor. The part of the file that is after the cursor, is pushed back against high memory. Thus, as much memory as possible is open, ready to accept your input.

You can visualize what happens when you issue a command to move, either in command or video mode: the amount you move and the direction determine how the data is pulled from either "inner end" of the file.

Let's take a simple example: a "C" command. "C" moves one character forward.

WM or ED just takes one character from the part of the file at the end of memory, and moves it to the end of the part of the file that is in the front of memory. A more simple editor with all the data packed at the front of memory, would be more efficient in that it would only move a pointer, and not any data.

However, the big "win" comes in when repetitively doing things that change the size of the file, such as changing all occurrences of multiple spaces, into a single space.

The command string would search for two spaces, and replace it with one, then move back one character. The "back up one character" means the command string will delete ALL redundant spaces. Without the backup, the macro would change 10 consecutive spaces to 5, not to 1.

On a 24K file, WM executed this macro in 15 seconds. MINCE, not having true command string ability, could handle the changing of two spaces into one, but couldn't then back up one character each time, so I didn't time it. PMATE and VEDIT apparently don't use the same techniques as WordMaster, since neither got even half-way through the file, in the three minutes I allowed them to run before interrupting them.

This is a worst-case example, and doesn't truly reflect the capabilities of the other editors. It surely should not be considered to be a criterion for selecting or rejecting any.

CONCLUSIONS

WordMaster's speed, compact size, and bug-free performance allow me to unhesitatingly recommend it to anyone looking for a reasonably priced, full screen, program editor.

However, this is only the first of four editors reviewed, so why not wait and see how the others stack up against WordMaster.

NEXT MONTH, I'll continue the reviews with the very delightful editor, MINCE, from "Mark of the Unicorn". It is as different from WordMaster, as night and day, yet exhibits some rather remarkable capabilities. They are not without some cost, though: MINCE is about three times as large as WordMaster, and requires a special "work file". In some ways, it is slower, and in others, faster than WM.

In future reviews, I'll discuss CompuView's VEDIT, Lifeboat's PMATE, and for that matter, any other appropriate program editors that fall into this class.

Since I have WordStar, I'll also comment on it briefly — not a complete review.

Vedit and PMATE are fundamentally similar to WordMaster, yet are quite unique entities.

The final wrapup part of the review will more thoroughly compare and contrast the editors.

I encourage your feedback.

More On dBASE II

Van Court Hare

Some further detective work on dBASE II reveals added clues to its internal operation, plus non-trapped "out of range" error conditions which can bomb dBASE II and CP/M. The following discussion provides the information needed to avoid problems and covers in turn: (1) variables (2) functions (3) the macros defined by &, and (4) several gaps and lapses in the dBASE II Manual and system.

dBASE II Variables

First, dBASE II uses variables of three distinct types (character, numeric, and logical) which may be stored in three distinct places (the Primary USE file, the Secondary USE file, and as "Memory" variables). Consequently, nine variable identification categories must be kept in mind, and any confusion of these may lead to error messages or incorrect results. dBASE permits the same variable name to be used in each of the three locations; that is, different variables with the same name, say X, may be in the Primary USE file, in the Secondary USE file, and also in the Memory variable area. The necessary distinction is by location, but the user is well advised to use different names for clarity, or to use a letter prefix to tag the distinction and avoid mixups. (Employ P.var for Primary, S.var for Secondary and Mvar [no period] for Memory variables, as suggested in the dBASE manual. The prefixes "P." and "S." have special meaning; the prefix "M" is optional.

File Field Variables

The exact name, type, and size of variables defined for USE files comes from the initial CREATE command, and accordingly little confusion occurs for variables so defined. USE file variables may be checked at any time with the DISPLAY STRUCTURE command for the file in USE. Variables defined by CREATE always reside in fixed-length fields, and dBASE II always works with such fixed length fields within fixed-length records in every data base (.DBF) file. This rigid, fixed format structure must never be disturbed unwittingly by a program or an entire database will be lost. Changes in file structure (MODIFY STRUCTURE, COPY, APPEND) must be done with extreme care and with backup for security. The current value of USE file field variables changes with the file record number in use. File structure for a given USE file is normally invariant unless purposefully altered by the user.

CAUTION: The maximum width of any file field variable is 254 characters (the maximum string length allowed by dBASE II), and the maximum total record size is 1000 characters, which must contain all record fields (up to 32 maximum). Both CREATE and MODIFY prohibit out of range specifications and provide clear error messages. APPEND correctly enforces the rules. However, duplicate field names in the same record are not detected by CREATE or MODIFY. If a record has two fields with an identical name, a reference to that name will process only the first match in the record structure; all subsequent duplicates are disregarded as if they did not exist.

The .DBF File Field Structure and .DBF Record Links

Because relational data bases are inherently organized in a fixed length record and field format for reference ease and processing speed, the user with limited disk space will want to size dBASE II's field and record lengths carefully, to avoid excessive storage of blanks in strings and of insignificant zeros in numerics. Users who attempt to process fields or records of widely different lengths must either pay a wasted disk space penalty or must abbreviate and truncate data to approach the ideal of tightly packed fields. Note that the dBASE II Pack command deletes records, not blanks (as is the case for this command in some variable-length, comma-delimited file processing systems). Preplanned within-record packing becomes the user's responsibility.

The field names used for relational data base file records form the links between .DBF files. Consequently, it is desirable to select field variable names with some forethought so that files can be tied together without unnecessary name conversions. The link name device can be used to remove the file constraints stated for dBASE II's standard file and also to build cross-reference links between files of different subject types.

To expand dBASE II files beyond stated limits, you can take one of two actions (or both) in succession. In what follows, the stated restrictions for one file remain. The scheme is to combine files to add more records to the same file structure, or to add more file fields to the same record key. The maximum field width (and maximum string length) of 254 characters must always be observed.

1—To expand the number of records in a file, use two files with an identical structure and put one file in PRIMARY and one file in SECONDARY. Process the PRIMARY file; and, at EOF, SELECT SECONDARY and continue processing. The sequence may be repeated and extended by bringing in one or more additional files with same structure. Change disks and RESET as necessary. By this means, file size limits are effectively removed for files with the *same* structure.

2—To expand record size, link records in PRIMARY and SECONDARY files that have *different* structures. Consider two cases: a) links on the same key, and b) links on different keys.

In case a) two files have different structures, but have a common "key" field (such as "Part:No") which appears identically in each file. Each file is also INDEXed on this key. Again, place one file in PRIMARY and one in SECONDARY, employ FIND to get the desired key item in PRIMARY, SELECT SECONDARY and FIND the same key. You now have sixty-two potential fields of data joined by the two common key fields. These fields may be processed by appropriate SELECTs to the group of fields desired. Repeated USE of additional files which contain the same key field can continue this expansion; consequently, the number of fields keyed to the same item is unlimited.

In case b), the procedure is the same as in a), but instead of a common key, you USE successive files, of different structure, linked on different keys. To illustrate, PRIMARY contains a file INDEXed on "Part:No" which also has a field "Vendor:No" in its structure. You FIND a desire part number, then consult the corresponding vendor number field. Now, if you place in SECONDARY a file of vendor data INDEXed on "Vendor:No" (a field which must exist in the SECONDARY file), you can employ the vendor number from the PRIMARY file to FIND the appropriate vendor in SECONDARY. You now have at hand the combined data for the desired part and for the associated vendor, and may process this data as desired (with SELECTs as needed). As before, the linking procedure just described may be extended by USE of further files. The necessary link to subsequent fields may be obtained from either the current PRIMARY or SECONDARY field, and so on until all the data desired have been obtained. Data you wish to capture may be STOREd in the Memory variable area temporarily, or processed directly.

Easy linking of the type just described is an important feature of relational data bases and of dBASE II's SELECT, USE, INDEX, and FIND commands. To supplement dBASE II's manual, which omits discussion of file design except for a few program sets, see S. Atre's *Data Base Structured Techniques* (N.Y., Wiley-Interscience, 1980, LC-80-14808, ISBN 0-471-05267-1). The proper design of file structure and linking mechanisms for a given application is the user's responsibility.

Memory Variables

dBASE II's Memory variables — which are defined by the command words STORE, ACCEPT, INPUT, GET — are allowed to be of *variable* length with a type which may change as the result of certain program statements.

(This internal flexibility of Memory variables is an important feature of the language. Memory variables permit intermediate storage of data as needed as well as any desired adjustment of data type and length which may be required to match .DBF file structures.)

Unfortunately, the DISPLAY MEMORY command shows neither a variable's length nor its type; only variable name and immediate value appear. As a result, the user must deduce Memory variable length and type by visual check, or by use of one of the language's functions, such as LEN (<string>). Consequently, it is not surprising that most of the confusion in dBASE II application arises from improper management of Memory variables. Memory variables are in no way altered by the record position of either the Primary or Secondary USE files, but only by program statements.

EXTREME CAUTION: The maximum number of characters allowed for the data of all memory variables is 1535 characters. To exceed this limit is fatal. Although a suitable error message "OUT OF MEMORY FOR VARIABLES" occurs for this condition, no recovery is provided, although dBASE shows a command level prompt. RELEASE has no effect. DISPLAY MEMORY will destroy the system and require a cold restart. Moreover, maximum string length of Memory variables is not checked and reported if excessive, leading to unexpected results. Suppose you have, say, string1 of 254 characters and string2 of 254 characters; an attempt to STORE string1+string2 TO string3 is superficially accepted, but what you get is string1 truncated to 252 characters in string3 without an error message warning. Accordingly, your command program can roll along wiping out characters and potentially destroying your data base. By all means test for maximum string length before big concatenation or blank squash operations. This is a true dBASE II bug to guard against.

(Incidentally, in the bug just cited, note that 254 + 254 = 508, and 508 −256 = 252. Because the maximum count which can be held in an eight-bit byte is 256 (0-255 decimal or 0-FF hex), the source of this defect probably lies in an accumulator overflow with no carry check. This same problem, as illustrated subsequently, arises for out of bounds substring functions.)

The Size and Type of Memory Variables

The type, size and content of Memory variables created directly by one of the commands STORE, ACCEPT, INPUT, or GET, and not copied from a .DBF record follow the type and size of the data provided. STORE 'Mary' TO MName yields a four-character string and STORE 5 to X gives a (two position) signed numeric with no decimal places. However all Memory variables - however defined - can be altered by certain subsequent operations. As shown in Exhibit A, concatenation and blank squash can increase the size of strings; functions such as VAL and STR can shift variable and data type, and the substring function can shorten string size. The user has the burden of keeping track of current Memory variable status so that (1) no string exceeds 254 characters, (2) total Memory variable size does not exceed 1535 characters, (3) no mixed type operations occur, and (4) no out of range reference is ever made to any string variable, as illustrated subsequently. This is not a trivial task, but crucial to success with dBASE II in Version 2.02. Note in particular the name, size, and type of both Primary and Secondary variables can change if you specify different USE files, and Memory variables change when you use RESTORE. Consequently, a command program which contains successive USE, SELECT, SAVE and RESTORE statements requires heightened caution in variable identification and application.

dBASE II Macros

Aside from variable housekeeping, the required use of macros (the &var construction) is the major difference between dBASE and BASIC. Put briefly, the dBASE II macro acts as a text-editing preprocessor which substitutes the data value of a named variable into a program command line before the line is phrased by the dBASE interpreter.

(continued next page)

The end result desired after macro substitution is an edited program command line which looks just like you had typed it on the console (this intermediate step is internal and invisible to the user.) Because you are editing text, the data value used by the macro must absolutely be of type string to be consistent with the other character text on the program line. Any violation of this rule causes an error message and halts the program. Further, the macro function &, which is exceedingly powerful and required for many desired actions, has specifications which are not sufficiently documented or illustrated in the dBASE Manual.

CAUTION: (1) Data used for macro substitution must be in string form; (2) The macro cannot exceed the overall 254 characters for string; (3) The delimiter for a macro &string is (a) a blank, (b) a carriage return, (c) an operation, or (d) another ampersand; and (4) after substitution of data invoked by the macro call, the resulting text construction cannot exceed the specific restrictions demanded for it by the dBASE II language. For example, variable names constructed by a macro cannot exceed ten string characters in length.

The Correct Use of Strings and Quotes in Macro Data

If a macro call is to return a 5, type STORE '5' to X (or use another form of string variable definition). Then, the computation ? &X + 5 will reduce to ? 5 + 5 and yield 10. If you STORE 'Address' to MFileA, then USE &MFileA will reduce to USE Address, the desired form. In the same way, if you want a quoted string, double quotes must be used in a literal STORE (or the data must contain surrounding quotes by other means). For example if you STORE "'$###,##' " TO Format1 (where the different quote types are necessary) &Format1 will give a quoted string as needed in @ 10,10 SAY Total:1 USING &Format1 which must reduce to @ 10,10 SAY Total:1 USING '$###.##' to meet the syntax requirements this statement. If you STORE '5' to MDog and STORE '6' to MCat, then ? &MDog&Mcat yields 56, but ? &MDog+&MCat yields 11, because "+" is taken as an addition operator. Using the same data, ? '&MDog'+ '&MCat' reduces to '5'+'6' (as text); an error, because strings cannot be added arithmetically. Finally, if you STORE "11" TO R and STORE "39" TO C, then @ &R,&C SAY "*" will display an asterisk at physical row 12, column 30 of your CRT (dBASE II counts from 0,0 for display). Expansions of the last illustration permit animated graphics, forms rules, bar charts, and similar art using the dBASE II language, but be careful with type conversion as you adjust R and C numerically.

A macro can be used to create subscripted variables and other language features you may miss. For example, suppose you have a file with the fields named Key1, Key2, Key3 ... Key9 and wish to index across these fields numerically, the equivalent of subscripting. Try this:

```
STORE 1 TO X
STORE STR(X,1) to Index
DISPLAY ALL FOR Key&Index = "Mary"
```

which will give all records in the current USE file which contain "Mary" in field Key1. Numeric adjustment of X will then select your choice of Key&-Index, a procedure which may be extended. (Remember, however, that after macro substitution, the resulting variable name cannot exceed ten characters in length.)

A traditional two-dimensional array can be created in a USE file by selecting the file column by the method just described, preceded by a selection of the row desired using GO TO N, where N is the row (record) number desired. In this process, the minimum row is 1; the maximum is 65535, or the maximum number of records currently in the USE file, whichever is smaller. Accordingly, the BASE of the array should be considered (1,1), not (0,0). This approach is slow, but does permit large arrays to be processed in the traditional way if desired.

Three Rules that Eliminate Trouble in dBASE II

The dBASE II command language was designed to promote structured programming to ease program maintenance, both objectives to its credit. However, dBASE II acts like a psychological ink-blot test for many persons familiar with BASIC and other high level computer languages. Because dBASE II keywords are identical or similar to those of BASIC, it is easy to conjure up fantasies of what dBASE should do — instead of what it does in reality. Heed the following command program construction rules to gain dBASE II's intended benefits.

1—Always take only one action, at its simplest possible level, on one line of a dBASE program. Avoid all complex constructions, particularly nested functions, substitutions in place if functions are used, and so on.

For example, this Microsoft BASIC-80 statement (which uses functions to increment a string, SCOUNT, by 1) nests functions and substitutes in place: 100 LET SCOUNT = STR$(VAL(SCOUNT) + 1). You may be tempted to translate to dBASE II as STORE STR(VAL (SCOUNT+ 1, 2) to SCOUNT to get the same result. The attempt will fail (as discussed later), because STR does not like substitution in place. To get the desired result, say increment SCOUNT from '9' to '10', do this:

```
STORE '9' to SCOUNT
STORE VAL(SCOUNT) + 1 to SCOUNT
STORE STR(COUNT, 2) to SCOUNT
```

2—Write dBASE II program lines so they exactly match the syntax form shown in Part II of the dBASE II manual, — without deviation of any kind. If in doubt, duplicate the exact form of the manual example which follows the syntax rule, taking care to match the data type and form precisely. In particular, assure that Memory variables used are of the proper type, and use literal values (or an equivalent macro) — not variable names — if a literal form is shows in a manual example.

3—Assume nothing! Many dBASE statements and functions which have the same name as in BASIC behave differently, and in some cases the dBASE II manual does not contain complete documentation, or is in error. If in doubt about any statement or function, construct a few test lines which use the questionable item or form and execute these lines one at a time in dBASE II's interactive mode with SET TALK ON. Record the actual results you get and be guided by reality, not preconception.

Some Critical Documentation on dBASE II Functions

All dBASE functions work as described in the manual with the exception of the string function "STR" and the substring function "$( )", for which the documentation is critically incomplete. Note however that VAL (unfortunately) differs from BASIC and always gives an integer result. INT also differs from BASIC's algebraic treatment and truncates with sign. dBASE II's STR format critically differs from BASIC's STR$ in that string length and decimal positions must be appropriately specified, whereas neither is required by STR$.

Moreover, the use of dBASE II functions is apparently subject to several foibles, including 1) unacceptable substitution in place, 2) out of range substring specifications which are not completely trapped, and 3) untrapped insufficient width specification for STR (as illustrated subsequently).

Certain functions, notable STR and VAL, appear to give erroneous untrapped results when used in place. See if you get this result:

```
. SET TALK ON
. STORE 1234.567 TO X
1234.567
. STORE STR(X,9,3) TO String
1234.567
. STORE STR(X,9,3) TO X
    0.000
```

The result 0.000 is an unreported error; the value of X is lost. Under some conditions VAL acts similarly.

```
. STORE "1234" TO X
1234
. STORE VAL(X) TO X
 0
```

Other functions place an extra leading blank in results processed in place:

```
. STORE "1234.5678" TO Temp
1234.5678
. STORE @(".",Temp) TO Temp
 5
```

Temp will, however, process correctly later, because leading blanks are disregarded in numerics. However, avoid trouble and never use functions in place.

Reasonable out of range references when using a substring function in dBASE II will be trapped and return a BEYOND STRING message. However, an excessively large out of range reference will give and store trash results, which at a subsequent step can incorrectly alter other variable and often bomb the system. To illustrate, try this, create and store and string, say, 214 characters to a memory variable called String. Next try successive values of <length> in

```
STORE $(String, 200,<length>) to
Templ
```

In this illustration, correct error trapping occurs for <length> values up to 57, but for 58 and over dBASE II performs the substring extraction erroneously, and stores to Templ <length> characters of garbled text — without warning. This is undoubtedly another accumulator overflow problem in dBASE with no carry check. Note that the attempt is to extract characters from 200 through 199 + 58 = 257, inclusive. Because 257 decimal is one more than the maximum 256 decimal count allowed in an eight bit accumulator (0-255 decimal or 0-FF hex) an overflow occurs. The carry bit is set, and the accumulator contains a value which passes the bounds test. Accordingly, the substring extraction looks okay to dBASE, which proceeds to do its thing with unfortunate results.

EXTREME CAUTION: An out-of-range error is relatively easy to make in the substring function, and because is is so critical it is worthwhile to compare <start> and <length> to LEN(<string>) in the program line prior to any substring action. In other words, program your own bounds check to circumvent this dBASE II error trap omission.

Be sure to provide adequate length value in STR. STR(<numeric expression>,<length>,[<decimals]>) requires that <length> and <decimals> (if that option is used) both appear as positive literal integer numerics (or an equivalent macro). The maximum specification of <length> must not cause the Memory variable area to be exceeded, nor exceed 254. The penalty for excess is potential wipeout. The minimum value for <length> is total field width, including sign, decimal point and any decimal positions specified. Violation of the minimum rule is treacherous, particularly if (1) the result of STR will be written to a database file, or (2) a macro depends upon a correct string form from STR for further action. Here is what happens: (1) If no length is specified, an error message results; (2) If <length> is of insufficient width for the sign (if any) and integer portion of <expression>, the field is filled with asterisks, but no error message is issued; (3) If <length> is insufficient for sign, integer portion, decimal place, and specified decimal places, the field is filled with asterisks to a few low order digits, but no error message is issued; and (4) If <length> is in fact sufficient for sign and integer portion of <expression>, but <decimals> is not wide enough to contain those available from <expression>, then the STR function truncates (not rounds) the least significant positions. Whether or not a full or partial field of asterisks will affect later program steps (where a string representing a number is expected) will depend upon later use and tests of the erroneous asterisk-containing result. The danger is that no messages occur for such errors, and a command program will not halt. Accordingly, portions of a database file may be incorrectly altered unwittingly by this route. The action of macros dependent upon an unexpected STR result is also uncertain and depends upon the application. Fortunately, if a macro is used to place an asterisk-containing string into a substring function for <start> or <length>, an error message will occur.

EXTREME CAUTION: Be sure the <length> specification in STR is big enough to contain sign, the largest integer ever to be encountered plus decimal point and any <decimals> specified. Because of the large number of type conversions required in dBASE II, errors in this function are likely.

dBASE II Documentation

The dBASE manual contains one or two minor errors and a few more underdocumented spots worth noting.

SET DEFAULT TO B: and DISPLAY FILES ON B: will not work unless you omit the colon.

The blank squash operation, totally undocumented, actually moves non-

(continued next page)

blank data to the left of a fixed field equal in size to the sum of all fields combined. See Exhibit A. This can lead to a resulting field of considerable width, containing mostly blanks, which rapidly eats up the 1535 character Memory variable area, or can exceed the 254 maximum string length, unless the blanks are stripped off or the variable RELEASEd. When dBASE II performs blank squash, a leading blank in a string is not squashed. Accordingly, "MARYbbbb" -"bANNEbbb" (where the lower case "b" indicates a blank) will yield "MARYbANNEbbbbbbb".

The screen-editing data acquisition construction:

@ <coordinates> SAY <expression> GET <variable> PICTURE <format>
(should appear on one line)

is very handy for data editing, but is tricky and only sparsely illustrated. To avoid problems and prior data on the screen, handle all data in string form, and convert to numeric later if necessary. Pre-define a string Memory variable exactly the width of the PICTURE, store blanks in it, and be sure to set it to blanks again after you have captured the data and moved it to another variable. You must use a quoted literal string for <format>, or a macro resulting in the same thing. Do no forget READ after the GET, or you will capture no data. If you do not reset <variable> to blanks, the current value of the variable will appear, and you can edit it in that form if desired. For initial data entry, the blanked-out variable form is best.

Finally, contrary to the documentation of SELECT, variables not in the current use file which are also uniquely named (are not duplicated between the Primary, Secondary, or Memory variable buffers) can indeed be altered by a REPLACE. Suppose you have X as a Memory variable, Y in Primary, and Z in Secondary, then SELECT PRIMARY. Now if you REPLACE Z WITH X+Y, dBASE II does exactly that. Duplicate variable names are protected: If X is in Primary and X is also in Secondary, then only X in the current USE file can be REPLACEd.

dBASE II appears (after many hours of testing) to have no quirks or bugs in the major file-handling commands, such as CREATE, MODIFY, DISPLAY, APPEND, COPY, file conversions and so on, and these utilities alone are worth the price of admission. Apparent errors which occur in REPORT and FIND can usually be traced to one of the quirks or misunderstandings previously discussed for the dBASE II command language, where most of the confusion and trouble occurs. With some appreciation of the design approach used in the associated command language, dBASE II is easy to use, fast, and a highly useful commercial product. More complete documentation and more graceful and complete error trapping and recovery procedures, however, are sorely needed for this powerful database system. For Version 2.02, the user is particularly advised to include programmed out of range checks for any operation which increases Memory variable length near the system's limits, to assure within-limit references in the substring function, and to provide sufficient width for STR conversions. Because any error message halts a command program and requires tedious reversion to a text editor for program repair, and because execution of carelessly created command programs can bomb the system, the user will save time if initial care is taken in command program construction to avoid errors in detail.

(*Editor's Note:* See the report in this issue on the new version of dBASE II.)

Exhibit A

```
***  dBASE II       Ver 2.02       4 May  81
. * Exhibit A -- Illustrates Blank Strip
. STORE "Mary       " to MName:1
Mary
. STORE "Anne       " to MName:2
Anne
. STORE MName:1-MName:2 to MName
MaryAnne
. ? LEN(MName)
  16
. * Find start of blanks less 1
. STORE @("       ",MName) to Count
  9
. STORE Count-1 to Count
  8
. * Convert Count to string for macro
. STORE STR(Count,2) to SCount
 8
. * Now strip blanks to right
. STORE $(MName,1,&SCount) to MName
MaryAnne
. ? LEN(MName)
  8
.
```

A Call For Manuscripts

Perhaps you've done some writing before. Or maybe you've always wanted to write. It could be that reading *Lifelines/The Software Magazine* has given you some ideas on what you have to contribute. We're interested in hearing what you have learned, and so are other readers. Whatever *serious* CP/M-80 compatible software you've been using, we'd like you to write for us. We like to publish both long essays and those short gems which can hold so much important information.

Send us a brief resume of your software experience, and samples of your previous writing, if you have any. (Don't be shy if you're *not* an experienced writer.) Then we can talk about your work and about payment for your efforts. If you wish to send actual articles, please submit in machine readable form; call first to make sure we can read your disks. Write or call: Editorial Dept., Lifelines Publishing Corp., 1651 Third Ave., New York, N.Y. 10028. Telephone: (212) 722-1700.

Gift Subscriptions

You should consider gift subscriptions to *Lifelines/The Software Magazine* for your friends and relatives who are involved in microcomputing. As you probably realize from your own experience, the price of a subscription is small for the money *Lifelines* can save you in a year. Just send a check or credit card number and fill out the form below*. (Or call [212] 722-1700.) We'll send your gifted one a note to let them know of their good fortune, *and* we'll send you a free Zoso T-shirt. (Don't forget to tell us your size.)

Your name and address:

Name ______________________

Address ______________________

City ____________ State ______ Zip ______

Shirt size ____________

☐ Check enclosed

VISA or MasterCard Number

Expiration Date

____

____

Signature (if payment is by credit card)

The name and address of the gifted one:

Name ______________________

Address ______________________

City ____________ State ______ Zip ______

*All orders must be prepaid by VISA, MasterCard or check. Checks must be in U.S. $, drawn on a U.S. bank. Subscription rates are $18 for twelve issues (one year) when the destination is the U.S., Canada, or Mexico. For subscriptions going to all other countries, the price is $40 for twelve issues.

LOUSY, CHEAP MACHINE!
WHAT'S WRONG NOW, JAMES?
I TYPED THE WRONG CHARACTER AND IT DID EXACTLY AS IT WAS TOLD!
CANT YOU FORGIVE IT? SHOW SOME COMPASSION!
IT SHOULD KNOW BETTER, MIKE! THIS ISNT THE FIRST TIME...
I THINK THE ROTTEN THING DOES IT ON PURPOSE!
GIVE IT A BREAK. AFTER ALL— IT'S ONLY HUMAN!
ZOSO
ROLLIN P. MAYER

The Osborne I Computer, Revisited

Kelly Smith

Upgrade

As promised in Lifelines December 1981, the 'Upgrade' to the Osborne I arrived. Despite what you may think, I am *not* on Osborne's payroll! This is just the best thing that has happened to personal computing in a long time, and so it goes.

Figure this one out: I return my Osborne I to the dealer (Guy Mongeau, 'Computers to Go', Westlake Village, CA) after notification that my upgrade has come in. Super, so at ten in the morning, I drop it off. They ask me if it would be any trouble to pick it up at two in the afternoon, not next week, but the same day! Here's the deal. . .no charge for the swapping-out of: (1) an entire new keyboard, (2) new ROM monitor chips, (3) a new CP/M system diskette, (4) a new WordStar diskette, (5) and complete checkout of the entire computer! Free parts, free labor, and all smiling as if they actually made money on the deal. . .strange!

All right, are the rest of you manufacturers and dealers of personal computers listening? Why can't *you* do as well to get low cost computing into the hands of the masses with support like this? Why is it that two Osborne I's could be purchased for the price of one of yours, and with better software 'to boot' (pun intended!)? Who else can respond in one day (actually four hours), and enthusiastically not make a dime? You had better start finding some good answers, 'cause Adam Osborne is going to blow you into the weeds if you don't!

So what is an upgrade? Well, first of all: fixes. The 'keyboard funnies' are now gone, with real 'N-Key rollover'. The keyboard 'speed' is excellent, and now includes full left/right and up/down 'arrow key' scrolling; and if you lose track of the display, you can 'zap' the screen immediately back to 'center stage' with Escape-[. Horizontal scrolling now automatically left justifies the screen as you type beyond the 52 character display width, a definite plus for BASIC or ASSEMBLY language programmers using long strings or comment fields (up to 128 characters per line!). And note that the 'arrow keys' and scrolling are optional for WordStar usage, giving added flexibility.

Well, this was as much as I had expected for an upgrade, just problem fixes. But wait, a good thing gets better. The Osborne I now supports five different printer interfaces, as well as ten 'Function' Keys (user programmable), a HELP program for the uninitiated user, 'autoload' of the WordStar and SuperCalc diskettes (as well as the HELP and CP/M 'boot'), for dummies that can't type 'WS' or 'SC', and (thank you Thom Hogan) an extended directory display utility. This is taken from CPMUG and alphabetizes the disk files, shows their size in 'K' bytes, disk file space used, remaining disk space and the number of active files. Overall system response is also improved by the 'unavailability' of the Real-Time-Clock; it's still there, but is *not* directly accessible/executable as it was in the previous version of the system configuration program SETUP.COM (i.e., the RTC is not 'hogging' the system instruction execution time by interrupt updates). And also, more documentation (missing some details, but I will explain that later on!), and a completely new manual promised in the first quarter of 1982. How do they do it?

The HELP Program

At this point, I want to wander a bit (or byte) and discuss the psychology of the Osborne I, the HELP file. I have been into microcomputing for about seven years now, and take a lot for granted when it comes to using a computer. They seem like simple mindless critters with very few mysteries left. I know CP/M well, and if a computer gives me trouble, I literally pound it into submission (my IMSAI needs an occasional kick to get it going!). The point is, I am not afraid of the things. But for the newcomer, IT (as in "will I hurt IT?") can be an awesome and formidable entity, not viewed as an extension of one's own mind.

Thom Hogan's (Osborne's Director of Software/Publications) 'autoload' HELP program gently introduces the newcomer to microcomputing; at an almost subliminal level. Soon the new user is actually pressing some keys, sounds dumb for you 'old timers', but I have watched many neophytes approach a computer keyboard with absolute fear! The HELP program 'talks' the user through the basics of the Osborne I hardware and software and is 'user friendly', informative (but not overwhelming), and just flat kinda cute. At the same time it teaches the user to 'get-the-feel' of the computer painlessly, very important, and doubly useful for a dealer demo'ing the hardware to a 'first timer'. Also impressive is that the graphics video display characters and attributes are utilized to 'show off', so to speak. Some of what the user initially 'sees and tinkers with' when HELP 'autoloads' is pictured at the end of this article. And here is a list of the HELP screens:

A	CP/M Assembler	N	MailMerge
B	CBASIC	O	Osborne Utilities
C	CP/M Commands	P	Printers/Printing
D	Diskette Handling	Q	Quitting Each Day
E	File Extensions	R	The RESET Button
F	Filenames	S	SuperCalc
G	Graphics	T	Testing
H	Programming Hints	U	CP/M Utilities
I	I/O Ports	V	External Video
J	Just Starting?	W	WordStar
K	Control Keys	X	Accessories
L	Layout of Memory	Y	Modem Connection
M	Microsoft BASIC	Z	Self Portrait

More Undocumented Features

Yep, more good surprises were in store for me with 'hidden features' *not* mentioned in the new documentation:

XDIR (eXtended DIRectory display) - will WRITE the current disk directory as a named file (any eight characters YOU wish to name it!) for archival purposes. As an example:

A>XDIR *.* -OSBORNE<cr>

does the directory display as well as create a copy of it as the file '-OSBORNE.DIR' on the 'A:' diskette. Or,

A>XDIR B:*.* B:-WSFILES<CR>

does the same thing but creates a '.DIR' file on the B: diskette.

"So what?", you ask. Take a look at newer CPMUG volumes; that leading '-' in the preceding the filename is not required by XDIR to create the filename, but is an 'identifier' for cataloging the diskette using Ward Christensen's CATALOG program from CPMUG Volume 25 or revised in Volume 70, totally compatible with the OSBORNE I. The usual procedure is to 'SAVE 0 -MYNAME.XXX' but XDIR will do it for you!

AUTOST ('autoload' HELP.COM) - is patchable to 'autoload' any .COM file and execute it at the system 'boot'! After you get tired of seeing the HELP file (and wait out the agonizing seconds to load 18K worth of text), you may want to just have the system directly bring in XDIR to see what's on your diskette. Of course, make a 'backup' of your system diskette (a BACKUP utility is provided on the new UPGRADE I diskette!), then follow along as we do a 'number' with DDT to get the Osborne I to do our bidding:

```
A>DDT AUTOST.COM<cr> <--- get 'AUTOST' into memory
DDT VERS 2.2
NEXT  PC
0900 0100
-D169<cr>
```

The last line 'dumps' memory starting from address 0168 Hex.

At this point you will see a digit '4' (the string length of HELP), ASCII text to 'autoload' HELP padded with seven 'null' bytes, a ASCII Control-Z (1A hex, to clear the screen), an 'ESCape g' sequence (1B Hex then 67 Hex, to set graphics mode) and a terminating ASCII '$' (24 Hex used to 'end' a message). So here's how you patch for 'autoloading' XDIR:

```
-S169<cr> <--- substitute starting at 169 Hex
0169 48 58<cr> <--- change 'H' to 'X'.
016A 45 44<cr> <--- change 'E' to 'D'.
016B 4C 49<cr> <--- change 'L' to 'I'.
016C 50 52<cr> <--- change 'P' to 'R'.
016D 00 .<cr>  <--- end substitution with 'period'
-S17B<cr> <--- substitute starting at 17B Hex.
017B 0D 1A<cr> <--- change "Loading..." message to
017C 0A 24<cr> <--- clear screen,'end' the message.
017D 0A .<cr>  <--- end substitution 'period'
-^C <--- Control-C to return to CP/M..
A>SAVE 8 AUTOST.COM<CR> <--- save 8 pp as NEW
                                 "autoload" XDIR program.
```

Now RESET the Osborne I, press the RETURN key, and the Osborne logo should briefly appear on the screen, and then the XDIR directory display. This will work for ANY '.COM' file, as long as the string length byte count is inserted at address 168 Hex, the proper ASCII hex values for the filename are used, and the filename itself does not overrun the 'null' character space...easy huh?

Keyboard characters '{' and '}' — for you PASCAL PHREAQUES that prefer to use 'curly brackets' instead of '(*' and '*)' for comment lines...use Control-, (comma) and Control-. (period), which maps to the uppercase equivalent of '<' and '>' characters on the new Osborne I keyboard.

ROM Diagnostic 'R ead in test' — Although it is now briefly mentioned in the HELP program that the built in ROM Diagnostic is available with keyboard Control-D at RESET, no mention is made at all of the 'R ead in test'...so here's what it's all about: entering 'R' at the keyboard allows inputing of 8 bit data via the RS-232 serial interface connector. The data is automatically stored starting at address 4000 Hex (while being simultaneously displayed on the screen), and incremented up through memory on a byte-by-byte basis determined by the first word (two bytes) initially received by the Osborne I's serial interface. That's why you will see 'Len=' (for record length) displayed; if you give the computer the LOW byte count, then the HIGH byte count (in binary!), it displays this value and then proceeds to 'gobble-up' all remaining incoming bytes as data, until the byte count has been completed. The data are then executed starting at address 4000 Hex as a program; just make sure that the data is executable 8080 or Z80 instructions, or strange and wondrous results will follow!

ROM Diagnostic 'M emory' test — This is documented, but leaves a lot to be desired; it only tests RAM from address 4000 Hex to EF00 Hex, with just a simple 'pattern stuffing/check for match' type of test, by rotating an initial pattern of A5 Hex through RAM. These means that only 'dead-duck' memory errors are found in the system, and that the entire bottom 16 kilobytes of RAM (where most of your programs will be running!) go untested. It would have been simple to relocate the test to address 4000 Hex (after testing that area) and then turn off the 'shadow PROM' and test the RAM from 0000 to 3FFF Hex. Maybe in the next version of the Monitor PROM(?).

Programmable Baud Rate Selection — for applications requiring changing the RS-232 serial interface Baud rate (300 or 1200 Baud) without having to run the SETUP utility to do the work, here is what's required, as to assembly language programming:

```
        ORG   4000H      <--- you must 'ORG'; we're going
                              to turn on 'shadow ROM' at
                              0000 to 0FFF range

ACIA$INIT:              ; reset ACIA, set Baud rate
;
        MVI   A,57H   <--- reset 6850 ACIA
        CALL  ACIA$CTL      serial interface
        NOP ! NOP ! NOP ! NOP <-- delay
        MVI   A,56H     <--- set 300 Baud('55H' for 1200)
        CALL  ACIA$CTL
        NOP ! NOP ! NOP ! NOP <-- delay so next we can
```

(continued next page)

```
                        look at ACIA status
    RET            <--- return fr/ACIA initialization
;
ACIA$CTL:          ; set-up ACIA control
;
    DI             ; disable interrupts
    OUT  0         ; turn off RAM, turn on PROM
    STA  2A00H     ; send to ACIA commnd/sttus prt
   (STA  0EFC1H) <--- do this if you want SETUP to
                   display actual Baud rate later
    OUT  1         ; turn on RAM, turn off PROM
    EI             ; enable interrupts
    RET            ; "Get back Loretta" (Beatles)
```

Well, that was easy enough, but how about something really useful like data communications between Osborne I's or even other systems (i.e., RCPM's (Remote CP/M Systems), Micro-Net, Forum-80, the SOURCE, etc.). If you have a handle on assembly language programming, then here is the 'hot tip': modify the 8K byte file MBOOT3.ASM from CPMUG volume 62 (either create a simple 'JAM-it-to-RAM' program from the serial port or type it all in with ED, gruesome, but you have to start somewhere!), to then receive the bigger (and better) PLINK1018.ASM from CPMUG Volume 62 (for the SOURCE) or MODEM926.ASM from CPMUG Volume 61 (for RCPM file transfers). Even better yet, buy it from me for $50 a shot for either PLINK or RCPMLINK and save some grief! It isn't cheap, but it does work. Send those checks to Kelly Smith, 3055 Waco Ave., Simi Valley CA 93063. I of course, expect to make millions from this!

O.K., so you want to do it yourself (sigh...) here are the routines that you patch (replace) in the CPMUG MODEM programs; and you figure out how to make the programs RUN with 'shadow PROM' in the Osborne I:

In addition to the to the ACIA$INIT routine, you need to send data OUT from the Osborne I, or receive data in to it:

```
OUT$ACIA:         ; send data TO the ACIA
;
    DI
    OUT  0
    STA  2A01H
    OUT  1
    EI
    RET           ; "Return to Sender" (Elvis...)
;
IN$ACIA:          ; get data FROM the ACIA
;
    DI
    OUT  0
    LDA  2A01H
    OUT  1
    EI
    RET        ; "Return to Forever" (Chick Corea)
;
ACIA$STATUS:   ; check for: XMTR buffer empty or,
               ;            REC  buffer full...
    DI
    OUT  0
    LDA  2A00H
    OUT  1
    EI
    RET            ; somewhere we've coded 1 more POP
                   ;than PUSH instruction..
                   ;"Return to Alpha Centauri"
                   ; (Kelly Smith...)
;
```

Now it's a matter of checking error status, transmitter ready status and receiver ready status, to send and receive data, so:

```
REC$CHAR:             ; receive a character from ACIA
;
    CALL ACIA$STATUS    ; get ACIA status
    MOV  D,A            ; save it...
    ANI  01H            ; mask off the REC bit
    CPI  01H            ; ready?
    JNZ  REC$CHAR       ; loop 'till it is...
    MOV  A,D            ; get ACIA status
    ANI  10H            ; framing error?
    JZ   REC$CHAR$1     ; if not, press on...
;
    do framing error handling here!
;
REC$CHAR$1:  ; no framing error,what about overrun?
;
    MOV  A,D            ; get ACIA status
    ANI  20H            ; overrun error?
    JZ   REC$CHAR$2     ; if not, press on...
;
    do framing error handling here!
;
REC$CHAR$2:  ; no overrun error, what about parity?
;
    MOV  A,D              ; get ACIA status
    ANI  40H              ; parity error?
    JZ   REC$CHAR$3       ; if not, press on...
;
    do parity error handling here!
;
REC$CHAR$3:             ; no errors...get the data
;
    CALL IN$ACIA        ; get data byte from the ACIA
;
'A' register says: "do with me what you may...
                    but respect me in the morning"
;
```

Fine, now we know how to RECEIVE data, let's send data out from the Osborne I:

```
XMT$CHAR:                    ; transmit character
;
     PUSH PSW                ; save chrctr in A on stack
XMT: CALL ACIA$STATUS        ; get ACIA status
     ANI  02H                ; mask off XMT ready bit
     CPI  02H                ; ready?
     JNZ  XMT                ; flags set by 'CPI'
     POP  PSW                ; get character from stack
     CALL OUT$ACIA           ; send the data byte...
;
;
```

SETUP: The Osborne I Configuration Program

I mentioned the addition of five different printer interfaces that can now be accommodated by the Osborne I, as well as the ten 'Function' keys; this is really a super addition to the functionality of the computer, but the documentation is less than clear in some areas. This should clear up some confusion:

Printers directly supported via the SETUP configuration are,
(1) the Epson MX 80 (with the serial card interface),
(2) the Qume Sprint 5 and 9,
(3) the Ricoh RP 1600,
(4) the Diablo 630 and 1620
(5) the Alpha Com DP 2000
(6) the Okidata Microline 80, 82A, 83A, 84, 2350, and Slimline
(7) the Olympia ES 100 RO
(8) all Centronics parallel interfaces
(9) all NEC serial interfaces
(10) the PET IEEE-488 printer interface.

Printers indirectly supported are (1) the IBM Selectric 50, 60, 75, and (2) the Olivetti 121, 221, and 321. These two printers are interfaced through Escon Products serial interface kit. Contact them at (415) 820-1256.

An impressive array of printer types! Minimal information is supplied however as to the RS-232 cable interfaces required... here is some help:

Osborne 1 'standard' RS-232	Epson, Qume, Ricoh, and Diablo
Pin 1 (Frame Ground)	Pin 1 (Frame Ground)
Pin 2 (Received Data)	Pin 2 (Transmitted Data)
Pin 3 (Transmitted Data)	Pin 3 (Received Data)
Pin 7 (Signal Ground)	Pin 7 (Signal Ground)
Pin 20 (Data Terminal Ready)	Pin 20 (Data Terminal Ready)

Osborne 1 'Standard'	RS-232 Okidata Microline
Pin 1 (Frame Ground)	Pin 1 (Frame Ground)
Pin 3 (Transmitted Data)	Pin 3 (Received Data)
Pin 6 (Data Set Ready)	Pin 6 (Data Set Ready)
Pin 7 (Signal Ground)	Pin 7 (Signal Ground)
Pin 20 (Data Terminal Ready)	Pin 20 (Data Terminal Ready)

Most important for all these printer interface capabilities is the ability to direct data to where you want it to go...and the folks at Osborne Computer let you redirect data to all the possible interfaces with a full implementation of the CP/M IOBYTE function, when more than one device type (printers usually) is being used. This is for the rich guys with two printers, who, running WordStar, want to use a Matrix Type printer (usually fast) for rough draft hard copy, and then send the finalized draft to a Daisy Wheel Type (usually slow) printer. Yep, it's true, the Osborne I can support any mix of two printer interfaces! Also, the protocols for communicating with these interfaces can be selected with SETUP for 'Standard' (hardware 'hand-shake'), 'ETX/ACK' handshake, or 'XON/XOFF' handshake. Although I have not tried, I suspect that by using the IOBYTE and 'XON/XOFF' protocol, you could communicate with large mainframe systems remotely with an acoustic modem, great for executives on-the-go, or students on a college campus. Just in passing, I have been using a Novation Cat modem to do file transfers to/from the Osborne I, and it works super!

As if all that was not enough, the SETUP program allows you to assign the numeric keys as function keys! Although the "User's Guide Addendum" says "...whenever you type the number key, the command will be issued", it should say "WHILE pressing the 'CTRL' key..."! A minor 'glitch' in the documentation, but I bet that it provokes a lot of phone calls to the Osborne Dealers with "It don't work!". It took me a few minutes to figure it out, but it does work. Also, when assigning a CP/M '.COM' filename to a 'Function' key, do it just as if you were executing the command at the CP/M system prompt; do not specify '.COM', and follow the command entry with a return, then two ESCape keyboard entries. This again is not clarified in the documentation, but then considering how fast the Osborne people are getting all this stuff into the hands of the purchaser, it's not surprising that some detail was left out. Thom Hogan is doing a superb job of it all.

I use XDIR on keypad '0', PLINK on '1', and RCPMLINK with a trailing 'T(erminal)' option on '2'. This makes for very fast command entry, and could just as well be set up for something like 'MBASIC STARTREK<cr>' if you want 'Instant On' STARTREK with one keystroke.

SYSGEN and MOVCPM

The UPGRADE I documentation goes into great detail explaining how to convert your original diskettes to the new requirements of the Osborne I, and don't try to cheat the sequence described by the way, or you will find out that it won't work! You must follow the ERAsing and PIPing exactly, to make room on the diskettes for the new diskette configuration. With all this done, I anticipated (with glee!) the moment for trying out MOVCPM and SYSGEN...Ooops!

Same old "SYNCHRONIZATION ERROR" problem as I described in the December 1981 Lifelines article! I called Thom Hogan at Osborne and suspect that he went scurrying off to the production area to find out what was going on, 'cause "It's supposed to work!".

Honest Thom, I must have called you fifty times after that to find out what the problem was, but the phone there is constantly busy! I was able to get the Osborne Operator (by calling after five P.M.) to take down the serial number of my upgrade I diskette that you wanted, so I hope she passed it on, sure would like to 'GEN' a smaller system, so that I can do some 'tricks' in high memory. But then as with everything else you have done to improve the machine, I expect that the 'FIX' is forthcoming. Also, just to comment a bit about the software, do you make it difficult to disassemble! The newer SETUP was pretty easy, but those guys at Sorcim made the original one an absolute devil to figure out. Not impossible, but definitely not easy; this is a commendation, not a complaint!

(continued next page)

"Help — G Screen"

THIS HELP OF INTEREST PRIMARILY TO PROGRAMMERS

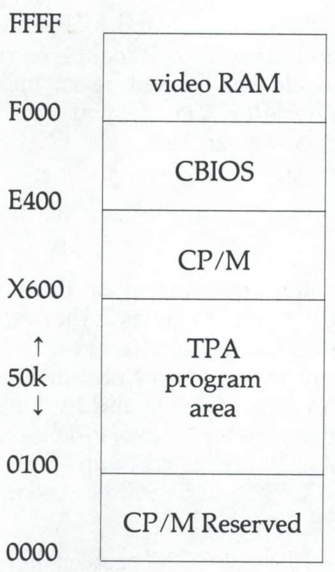
The Osborne I is capable of displaying a number of special characters. The following character sequences invoke each special display mode:

start underline	<ESC> 1
stop underline	<ESC> m
start half intensity	<ESC>)
stop half intensity	<ESC> (
start graphics	<ESC> g
stop graphics	<ESC> G

graphics character set: [the sample graphics set appears here!]

To print the graphics you see on the screen you'll need a printer that has graphics capabilities and software to translate the display to the printer.

"Help — L Screen"



In addition to the main bank of RAM shown at the left, there is a shadow bank, where the ROM & I/O ports reside. For more details, See the appropriate chapter in manual.

MEMORY USAGE:

WordStar	44k
MBASIC	24K
CBASIC	18k
SUPERCALC	40k
UTILITIES	16k

"Help — T Screen"

Your Osborne I has a built-in diagnostics program. To use it, press the RESET button, then type a control-D. You may then:

D — DISK TEST: use a brand new diskette that you've just formatted.

K — KEYBOARD TEST: type characters and watch the monitor to make sure they're displayed.

M — MEMORY TEST: continuously tests your computer's memory. The changing character at the top of the screen is normal.

If you find the test messages cryptic or don't understand something, have your Osborne dealer run the tests for you.

"Help — X Screen"

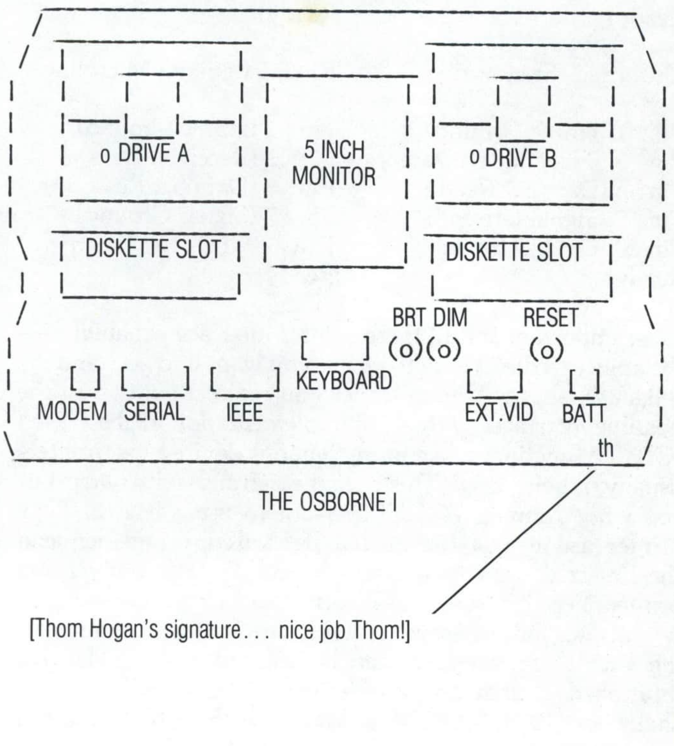
Accessories planned for the Osborne I*
- 12″ display monitor
- double density disk option
- battery pack for portable operation
- direct connection modem for communications
- Osborne Authorized software releases

*Contact your dealer for information regarding availability of these items.

"Help — Y Screen"

It is possible to use your Osborne I to transfer information via the telephone if you have bought the optional "modem" accessory with your computer. The modem plugs into the connector at the far left bottom of the Osborne I, with the other end to be connected to your telephone jack or telephone handset (depending upon which you purchased). A full description of how to use your modem is provided with it. Do not lose this information as it is not repeated in the Osborne Reference manual.

"Help — Z Screen"



[Thom Hogan's signature . . . nice job Thom!]

Tips and Techniques

Ron Fowler has sent us this tip on conditional interrupts in the 8080 and Z80 CPUS.

"Quite often, in the course of writing interrupt-driven programs, it is necessary to disable interrupts around certain indivisible blocks of code. When such a block has finished executing, interrupts can be re-enabled. I've often found it necessary to enable interrupts ONLY if they were enabled on entry; if they were not enabled, then my program must not enable interrupts.

This is difficult at best with an 8080. The requirement is that ANY program in the system that unconditionally enables interrupts must set a special flag as a reminder that interrupts are on. This flag must then be tested before interrupts can be safely re-enabled. Some sample code:

```
MYSUB:  DI                   ;disable the interrupts

   <non-interruptable code>

        LDA     INTFLG       ;check if interrupts can be enabled.
        ORA     A
        RZ                   ;return if they cannot
        EI                   ;otherwise enable interrupts
        RET
```

A program that unconditionally enables interrupts must ALWAYS do

```
        MVI     A,1      ;set interrupt flag
        STA     INTFLG
        EI
```

This is especially tough when many of the programs you're using are not available in source form (e.g., DDT and SID, the CP/M debuggers). The Z-80, however, allows the interrupt flag to be tested at any time. Whenever the interrupt or refresh registers are loaded into the accumulator, the interrupt status of the CPU is copied into the parity flag. This allows the conditional EI to be performed with no dependence on any kind of external flags. This technique is illustrated in the following sample subroutine, given in both TDL and ZILOG assembler code:

```
   -- TDL --             |      -- ZILOG --
                         |
MYSUB:  LDAR             | MYSUB:  LD     A,R     ;fetch refresh reg
        PUSH    PSW      |         PUSH   AF      ;save parity flag
        DI               |         DI             ;now disable interrupts
                         |

             <non-interruptable code block>

        POP     PSW      |         POP    AF      ;retrieve parity flag
        RPO              |         RET    PO      ;retn if int ff was off
        EI               |         EI             ;it was on, restore ints
        RET              |         RET
```

CPMUG Volume 78, Catalogue

CPMUG Volume 78

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.078	CONTENTS OF CP/M VOL. 078
	8K	ABSTRACT.078	Abstracts of programs
	5K	U-G-FORM.LIB	Users Group submission form
	8K	VOLUME78.DOC	DOC on programs not having some other form of documentation.
078.1	4K	/.ASM	Quickie SUBMIT from command
078.2	1K	/.COM	line.
078.3	1K	/DUP.COM	Same as /.ASM but produces a second copy of $$$.SUB to facilitate re-executing
078.4	10K	BMAPORIG.ASM	Prints 2.2 disk allocation
078.5	1K	BMAPORIG.COM	bit map
078.6	2K	CRCK.COM	to check files on this disk
078.7	2K	CRCKLIST.078	File of all file CRC's
078.8	3K	D.COM	Type D to see if any files have been lost on this disk
078.9	10K	DUH.COM	Disk Utility for the H/Z-89
078.10	3K	DUH.Z80	Source for DU relocator
078.11	51K	DUU.ASM	Disk Utility Universal, works
078.12	7K	DUU.COM	with 1.4 and 2.2
078.13	14K	DUU.DOC	DOC on above
078.14	8K	EPROM.ASM	SSM PB1 PROM prog. rtn.
078.15	2K	EPROM.DOC	DOC on above
078.16	4K	FMAP.COM	File map updated for 1024 dir entries and CP/M 1.4 or 2.2
078.17	1K	IF.COM	Continue or abort a SUBMIT
078.18	2K	LPRINT.ASM	Paginated file output to LIST
078.19	12K	MAKE.ASM	Make a file a different user #
078.20	3K	NOTATE.ASM	Add comments to an .ASM file
078.21	1K	NOTATE.COM	.COM of above
078.22	3K	PATCH.ASM	Patch CP/M to show user: "A0»"
078.23	1K	REPEAT.COM	Repeat a SUBMIT "nn" times.
078.24	10K	SDCOPY.ASM	Single-disk file copy program
078.25	1K	SDCOPY.COM	' '
078.26	3K	SDCOPY.DOC	' '
078.27	14K	SWAPCOPY.ASM	Another single disk file copy
078.28	2K	SWAPCOPY.COM	program
078.29	3K	SWAPCOPY.DOC	' '
078.30	1K	TestProt.BAS	Sample to test UN.COM
078.31	5K	UN.COM	Unprotect MBASIC programs
078.32	23K	XREFPRN.ASM	Prints a cross-reference from
078.33	3K	XREFPRN.COM	a PRN file to CP/M LIST device

. . . And Abstracts

Abstracts

ASM-COM/DUP.COM
.ASM by John M. Kodis, reads a command line which may contain several program invocations. A $$$.SUB file is built with these commands. This file is then executed. This is functionally equivalent to /.COM on CPMUG vol 40, for which I did not distribute the source because it had other functions in it I was not prepared to document.

A compile-time option allows writing a second copy of the $$$.SUB file, so you may "ren $$$.sub=/.sub" then ↑C, to re-execute the same commands. A version compiled with this option is on the disk as "/DUP.COM".

BMAPORIG.ASM/COM
(Caution: 2.2 only)
BITMAP for CP/M 2.0+ by Lauren Guimont. The bitmap idea is based upon my ALLOC program which was hard-coded for single-density disks only.

This version prints total disk capacity, amount used, and amount left. The "amount used" is handy to have when wishing to know if the files on a DD disk will fit on a SD disk.

CRCK.COM
by Keith Petersen.
The program included on every CPMUG volume to allow you to verify the contents of the disk. Executing "CRCK *.* F" writes CRCKLIST.CRC. That file has now been renamed to CRCKLIST.078 so CRCK *.* F won't inadvertently overwrite the original file.

D.COM
by Ward Christensen
Just type "D" to verify that all the files that are supposed to be on the disk, are. There is also a generally useful program to maintain disks–deleting unwanted files, etc. "D H" prints help. Source to be released on a future disk. Most significant new feature is "D CU" option for cleaning up a disk.

DUH.COM/Z80
Bill Norris wanted to run DU (CPMUG volume 46) on a Heath/Zenith-89. He made a relocator for it. The .COM file includes the relocator, and DU itself. It runs at either 100h or 4300h. You would of course have to figure out how to transfer these files to a H/Z-89 system.

DUH.Z80 is NOT the DU source. It is a relocator for the DU version from CPMUG volume 46. I presume it could be modified to work with the newer DUU on this disk, perhaps by changing the load addresses since DUU is quite a bit longer than the old DU.

NOTE that it is written for a Cromemco Z-80 assembler, yet generates only "8080" instructions. The program self-relocates(!) itself. The source was mainly supplied to satisfy the curious. See *Lifelines*, Volume I, Number 1 for information on using DU. Also, some, but not all, information from DUU.DOC (next) applies.

DUU.ASM/COM/DOC
This is an enhancement of the DU (Disk Utility) program from CPMUG Volumes 40 and 46. Ron Fowler made it "universal" in that it looks at disk parameter blocks to run on virtually any 1.4 or 2.2 disk system. I added a few bells and whistles, and modified it ("removed stylistic changes") to make it most closely reflect what had been NECESSARY to change from those earlier CPMUG versions.

The source contains equates for non-0 CP/M systems; I do not know if it ACTUALLY works on them. Perhaps someone will report back?

See .DOC for complete details.

EPROM.ASM/COM/DOC
This is Thomas Sly's EPROM programming program for the Solid State Music's PB1 PROM programming board.

The program runs under DDT along with the software to be burned into EPROM. Includes 'copy','verify' and 'program' of 2708, 2716 and 2516 type EPROMS.

Note this program has a 4000H byte open area in it at the BEGINNING so if you are tight on disk space, you may want to buy back 16K by simply reading in what you want to burn at a bias, rather than reading it at 100H.

FMAP.COM
My update of CPMUG Volume 40 FMAP. The reason this is on the disk is that I get more letters about FMAP not working than about any other program. The earlier versions didn't support enough directory entries, and didn't handle 2.2 "80H" bits in the file names.

This version supports 1.4 and 2.2, up to 1024 directory entries, finds files in all user numbers, etc. The bit map function has been updated to handle 2.2 also.

It still needs some work to properly handle disks with >2K allocation groups. I believe only the "K" option doesn't quite work with those large allocations typical on large hard disks.

There is insufficient room left on this volume to include the source; I will continue working on it, and place it on a future volume.

Further DOC via "FMAP H" command.

IF.COM
My program, used in SUBMIT to test if a file does or doesn't exist.

LPRINT.ASM
by P.P.H. Lee, Royal Melbourne Institue of Technology.
This program is a substitute for STAT LST:=LPT: and STAT LST:=TTY: Its format, LPRINT ON and LPRINT

(continued next page)

OFF, is easier to remember.

MAKE.ASM
Terry Lewis wrote this nifty program to change the user number of any existing file or set of files. This makes the CP/M USER function much more usable, since files may quickly be changed from user to user without having to PIP them.

NOTATE.ASM/COM
Martin E. Nason updated this program, originally CPMUG program 29.13. Those of you who feel 13 is unlucky can use this for an example. Apparently 29.13 got badly screwed up somehow – control chars in the file, etc. Martin took the glitches out, and sent this version in. It requires many of the macro libraries distributed with MAC. A .COM file is supplied, and works well. There should be no need to reassemble it. It is also a good sample program of the use of the various Digital Research MAC libraries.

PATCH.ASM
Terry Lewis submitted this program to print the current CP/M 2.2 user # as part of the "A>" prompt of CP/M, e.g. "A0>"

REPEAT.COM
For SUBMIT, this allows repeating a SUBMIT file "nn" times. Look at VOLUME78.DOC for details.

SDCOPY.ASM/COM/DOC
Don Bailey's revision of CPMUG 29.30, a single-disk MOVE program, which only handled files that fit in memory. This version handles larger files. Further information is in SDCOPY.DOC.

SWAPCOPY.ASM/COM
by John M. Kodis. This is a file transfer utility for single drive systems. It was designed to allow a single drive system to be used to transfer any file or group of files from diskette to diskette. There are no restrictions on the number or size of the files. Further information is in SWAPCOPY.DOC.

XREFPRN.ASM/COM
Version of CPMUG 8.27, modified to produce a cross-reference from the output of ASM. Modifications by P.P.H. Lee. Like its predecessor, it goes directly to the CP/M LIST device, with numbered lines, then produces a cross-reference by line number (not address). This does not appear to be based upon CPMUG 36.36, which was another derivative of 8.27.

UN.COM
Bill Norris' program to unprotect MBASIC version 5.x programs. This single .COM file contains its own documentation. A bit is available via typing "UN HELP", and even more by doing "TYPE UN.COM". Yes, that's ".COM". He did some clever things. This program looks "very nice" in that you pre-load it into your system, and it "goes somewhere where it doesn't take TPA room". Then, you load MBASIC. Load your BASIC program, and if you type ↑U it becomes unprotected. Voila! The author suggests that people use it "...for the recovery of their OWN programs."

Ward Christensen

Ordering From CPMUG

Many of you have inquired about ordering volumes from The CP/M Users Group.

The complete catalog of volumes is available for $6, prepaid, to the U.S., Canada and Mexico. The price is $11 for catalogs sent to all other countries.

Software is obtainable exclusively on diskette; CPMUG does not supply printed documentation. Prepaid media and handling charges are as follows:

8″ IBM format to U.S., Canada, Mexico $8

8″ IBM format to all other countries $12

North Star format to U.S., Canada, Mexico $8 or $12

North Star format to all other countries $12 or $16

As you will note, software is available in 8″ IBM and North Star 5¼″ formats. Write for a price list of North Star format volumes, as prices vary for volumes in this format.

Payment covers the cost of the diskette(s), packaging, and shipping. Checks must be in U.S. dollars, drawn on a U.S. bank. Domestic shipping is via UPS where a full street address is given; all other orders are via U.S. Postal Service.

CPMUG receives orders by mail only; they have no phone service. Orders should be sent, with prepayment, to CPMUG, 1651 Third Ave., New York, N.Y. 10028.

Members receiving the material are reminded that software contributions are necessary if the exchange program is to prosper. Software contributions are gladly received for inclusion into the Library with the understanding that the contributor is authorized to make the material available to others for their individual non-commercial use.

Software should be accompanied by sufficient documentation in the form of internal comments or accompanying *.DOC file to permit the material to be applied and/or modified. Where appropriate, the documentation should describe any supporting software (interpreter, memory, clock, etc.) necessary to use the routine. For your convenience, a comprehensive submittal form is now included on most distributed diskettes.

Contributors are invited to request any Library Volume in exchange for the one submitted.

STOP

...what you're doing right now if your subscription began last April. You're about to miss an issue.

If your subscription began last April you've been notified that this is your last issue. If you haven't responded, get out that renewal form and drop it in the mail right now with your payment. Or get out your VISA or MasterCard and call *Lifelines/The Software Magazine* Subscription Dept. at (212) 722-1700. The address is: 1651 Third Ave., New York, N.Y. 10028.

Configuration Parameter Storage for PL/I-80

Michael J. Karas

Many microcomputer applications programs require that certain parameters of the program execution environment be variable, so that the program can be tailored to the task at hand while retaining a means for keeping the parameters semipermanent. Retaining the configuration information saves re-entering or re-installing the configuration parameters each time the application program is loaded into the user's computer memory. An often-used "normal" method of retaining the parameters is to store them in a small separate read/write disk file. The side file has its problems, however, since it must be kept on the disk as part of the applications system.

Another method exists that provides a particularly elegant solution to the parameter storage problem. Examples of this method include the CRT Terminal characteristics storage contained inside the object code image of MicroPro's WordStar or Ashton-Tate's DBASE II. In these cases, the parameter information is accessible from the same file as the real, executable program. The method of configurable parameter access and update presented in this article is devised to permit the storage of the parameters within the first logical sector of the object code file, namely the first 128 bytes of the CP/M-80 COM file.

Assembly language interface and coding examples of external routine access within the Digital Research PL/I-80 applications programming environment were presented in a former two part *Lifelines* article entitled 'Assembly Language Interface for PL/I-80'; it appeared in the September and October 1981 issues (Volume II, Numbers 4 and 5). The parameter mechanism shown here utilizes the same type of assembly language interface, so the mechanization details will not be repeated here.

The heart of the configuration data storage is a small module of 8080 assembly language that performs the primitive functions of accessing the first (base) logical sector of the appropriate disk file and allows movement of the data to the 'host' PL/I-80 program. The following paragraphs will discuss the philosophy of the base sector access method, and operation of the various assembly language routines.

The design of the assembly language modules below has been done in a way to allow the reader to easily grasp the concept. Many other coding formats or interface conventions could be devised to adapt the scheme to one's application in either a more diversified or more efficient manner. A short discussion of the modification possibilities is given further on.

The base sector scheme assumes that a PL/I-80 program is to be linked to the code shown in Listing 1. The assembly language would be linked first, so that the buffer is forced to reside within the bounds of the first sector of the resultant object code file. The base sector of the file name given in the file control block (FCB) could then be read and written to access the appropriate parameter data. Note that as coded, the FCB can contain the name of any disk file on the default logged drive, including the name of the COM file that contains the assembly code. If the name is different from the loaded object program, then that program is most likely an installation or configuration program that "sets" the operating parameters of the differently named file. If the name is the same as the loaded command COM file, then the program is able to access information that is a direct part of itself. There is absolutely no requirement that the name of the file be anything specific.

The assembly language program contains four entry points, RDBASE, WRBASE, GETPARM and PUTPARM. These respectively read, write, fetch and place data to the base sector buffer contained as a part of the host program. The method used by the assembly language to allow writing of the base sector without a subsequent file 'CLOSE' operation requires that entry point RDBASE be called first. This opens the file, establishes the file control block parameters and reads one logical sector. The WRBASE entry point may then be called to force rewrite of the sector with new data. The tricky FCB manipulation done in the WRBASE code sequence has been shown to work for Digital Research's CP/M-80 Versions 1.4 and 2.2. Other versions may be excepted and the method to force sector rewrite for other CP/M-80 look-alike systems may be different.

The memory map of the final linked object code file is shown below. Note that due to the initial JUMP instruction installed at address 0100H by the link process, a total of 125 bytes (one 128 byte logical sector minus a 3 byte JUMP) are available to be read from the appropriate disk file by the RDBASE assembly language entry point.

Memory Address	Contents
0100H to 0102H	Jump to start up initialization point inside of PL/I-80 program inserted at Link time.
0103H to 0182H	128 byte buffer to read base sector accessible disk file into.
0183H to top of assembly code	Object code position of machine language entry points defined by program module shown below.
top of assembly code+1 to end of PL/I-80 object image	Object code image position of PL/I-80 applications program.

(continued next page)

Listing 1 is the program in 8080 machine language that provides entry point definition for a PL/I-80 program to access the base sector of a disk file.

The list below includes some of my ideas on how the previously presented assembly language structure could be changed to fit other applications. Use these as guidelines to jog your memory and challenge your programming expertise as you modify the code presented here.

a) The entry points RDBASE and GETPARM could be combined into one function. This eliminates one CALL in the example PL/I-80 program shown below - i.e., the base sector would be read and moved to the host program all in one shot.

b) Like in a) above, the entry points PUTPARM and WRBASE could be combined into one function. As before, this would eliminate one CALL in the applications program and the base sector data would be moved and written all in one operation.

c) The need for the buffer could be eliminated if the RDBASE and WRBASE entry points were called with a pointer variable defined as the base storage of the PL/I-80 program data area. The system call to set the read/write buffer address (SET DMA ADDRESS) would use the passed pointer value as the host disk access buffer address.

d) The buffer could be retained but the need for internal program storage allocation eliminated if the RDBASE entry point returned a pointer to the applications program telling of the location of the buffer. This would permit the applications program to overlay the data access structure directly over the buffer. The pointer returned would be used in pointer qualified references to the overlay structure or array. Note that both ideas c) and d) eliminate the need to perform the 125 byte block moves between the local buffer and the internal PL/I-80 parameter storage area.

e) A variable nature could be attributed to the file access entry points if the PL/I-80 program passed the file name to the assembly language entry points.

f) A final suggestion would be to devise a second, slightly simpler assembly language module that didn't read the COM file at all but assumed that the data was already in memory as a result of the applications program being loaded by the CP/M-80 command processor. In this case it would be assumed that another separate "installation" or "configuration" program would write the parametric information into the base sector utilizing the scheme given in this article.

Given at the end of this article in Listing 2 is the shell of a PL/I-80 program that could use the entry points of the already-defined base sector routines. The most significant feature of this program is the use of based storage allocation for the parameter area. This permits the data from the parameter area to be accessed in an application-dependent fashion. One declare defines a complete 125 element array of single byte characters. Second and third declares provide a simple illustration of two other overlay access templates which allow reference to data variables from the parameter sector. The example declares give only an inkling of the possibilities available. The programmer should be careful to make sure that the memory storage requirements of any overlay template do not exceed the 125 byte maximum. (This warning may be ignored by the more experienced PL/I-80 programmer as long as ALLOCATE statement given in the program names the largest item of those pointed to by pointer variable P).

Several special notes about this program are discussed below:

a) The program utilizes implied pointer references. The programmer is free to use pointer qualified references of fixed storage allocation for the various data aggregates. In the latter case the pointer for passage to the GETPARM and PUTPARM subroutines could be determined by the ADDR built in function.

b) Note that the program contains declare statement entries to define the external assembly language entry points. These could be placed in a separate .DCL file and included into your PL/I-80 source program with the %INCLUDE statement.

c) The shown order of external entry point calls must be maintained for the assembly language to function properly. The GETPARM and/or PUTPARM functions may be called as many times as desired to move data into and out of the buffer. The RDBASE function may be called at any time, in a multiple manner if desired. WRBASE may spoil the validity of the default disk drive if the entry point is called before the RDBASE function. Finally, the WRBASE function may only be called once after each RDBASE call. (At your own risk if you don't use genuine CP/M-80 1.4 or 2.2 and follow the rules here).

In order to get the base sector parameter routines positioned properly to match the memory map previously shown, the following link sequence is used. The Assembly program assumed to be named "PARM.ASM" is converted to a .REL file with the Digital Research relocating macro assembler, RMAC. The PL/I-80 program named "TESTPROG.PLI" is compiled with the PL/I-80 compiler into a relative format .REL file. Both are appropriately linked and formed into "TESTPROGCOM" with the command:

```
A>LINK TESTPROG=PARM,TESTPROG<cr>
```

where all files are assumed resident on logical CP/M-80 disk drive A:.

In the normal manner I hope the example presented here provides the reader with a method to solve some lingering problem. Although the program samples are short and probably easy to type in, I am still happy to provide the complete source for this article. If you want a copy, just send me a BLANK pre-formatted eight inch single density, single sided floppy diskette, in appropriate packaging, to the address shown in the program listings. Please include a pre-postage paid return mailer envelope or sufficient return postage. Avoid sending CASH. Note also that it now costs about a dollar and a half to send a diskette, due to recent postal rate increases.

[See Listings 1 and 2 on the following page.]

Listing 1

```
;*************************************************************************
; MICRO RESOURCES ASSEMBLY LANGUAGE PARAMETER SECTOR ACCESS PROGRAM
;*************************************************************************
;
;       THIS MODULE OF SOFTWARE IS A SMALL PROGRAM THAT IS TO
;       BE LINKED TO THE BASE OF AN APPLICATION OR PL/1-80
;       PROGRAM. THIS MODULE CONTAINS ENTRY POINTS THAT:
;
;         A) READ IN THE PRESENT CONTENTS OF THE FIRST 128 BYTES
;            OF A FILE NAMED AS DETERMINED BELOW. BUFFER IS IN THIS
;            MODULE.
;
;         B) WRITE OUT THE PRESENT CONTENTS OF THE LOCAL 128 BYTE
;            BUFFER TO THE FIRST SECTOR OF THE FILE NAME DEFINED
;            BELOW.
;
;         C) COPY LAST 125 BYTES OF THE BUFFER TO AN AREA OF MEMORY
;            POINTED TO BY THE ENTRY POINTER.
;
;         D) COPY 125 BYTES FROM SOME AREA OF MEMORY TO THE BUFFER
;            POINTED TO BY THE ENTRY POINTER.
;
;
;       WRITTEN BY:
;       MICHAEL J. KARAS
;       MICRO RESOURCES
;       2468 HANSEN COURT
;       SIMI VALLEY, CA 93065
;       (805) 527-7922
;
;**************************************************************************
;
;
;SYSTEM INTERFACE EQUATES FOR FILE I/O AND ERROR MESSAGE PRINTING
;
BDOS    EQU     0005H               ;FILE MANAGER ENTRY POINT
PRINT   EQU     9                   ;PRINT STRING FUNCTION
OPEN    EQU     15                  ;OPEN FILE FUNCTION
READR   EQU     20                  ;READ RECORD FUNCTION
WRITER  EQU     21                  ;WRITE RECORD FUNCTION
STDMA   EQU     26                  ;SET DATA BUFFER ADDR
;
        NAME    'PARM'
        TITLE   'PARAMETER SECTOR ACCESS MECHANISM FOR PL/I-80'
;
;
;DEFINE PUBLIC ENTRY POINTS FOR THE FUNCTIONS IN THIS MODULE
;
        PUBLIC  RDBASE              ;ENTRY TO READ FIRST SECTOR OF FILE
        PUBLIC  WRBASE              ;ENTRY TO WRITE FIRST SECTOR OF FILE
        PUBLIC  GETPARM             ;ENTRY TO TRANSFER 125 BYTES OF
                                    ;..PARAMETERS TO PL/I-80 PROGRAM
        PUBLIC  PUTPARM             ;ENTRY TO MOVE 125 BYTES OF
                                    ;..PARAMETERS FROM A PL/I-80 PROGRAM
;
;
;SETUP BUFFER AREA FOR FIRST SECTOR OF FILE
;
BUFFER:
        DS      128                 ;LOCAL SECTOR STORAGE AREA FOR
                                    ;PARAMETERS 125 BYTES REFERENCABLE
;
;
;OBJECT FILE ACCESS FCB
;
FILEFCB:
        DB      0,'TESTPROGCOM',0,0,0,0
        DS      16
        DB      0
;
;
;POINTER PARAMETER FETCH SUBROUTINE
;
GETPNT:
        MOV     E,M                 ;GET LOW BYTE ADDRESS
        INX     H
        MOV     D,M                 ;HIGH BYTE OF ADDRESS
        XCHG                        ;(HL) = ADDR OF POINTER STORAGE
        MOV     C,M                 ;LOW BYTE OF POINTER TO (C)
        INX     H
        MOV     B,M                 ;HIGH BYTE OF POINTER TO (B)
        RET
;
;
;ENTRY POINT TO READ FIRST SECTOR OF FILE
;
RDBASE:
        LXI     D,FILEFCB           ;POINT TO FILE FCB
        MVI     C,OPEN              ;TRY TO OPEN FILE
        CALL    BDOS
        INR     A                   ;CHECK RETURN CODE
        JZ      RDERR               ;PRINT ERROR BEFORE RETURN
        LXI     D,BUFFER            ;SET DMA ADDRESS FOR READ
        MVI     C,STDMA
        CALL    BDOS
        LXI     D,FILEFCB           ;POINT TO FCB FOR READ
        MVI     C,READR
        CALL    BDOS                ;GO READ IT IN
        RET
;
RDERR:
        LXI     D,NOTPRES           ;POINT TO NOT PRESENT ERROR MESSAGE
        MVI     C,PRINT
        CALL    BDOS
        RET
;
NOTPRES:
        DB      0DH,0AH,'EXECUTABLE FILE NOT PRESENT ON DEFAULT DISK'
        DB      0DH,0AH,'$'
;
;
;ENTRY POINT TO WRITE FIRST SECTOR OF FILE
;
WRBASE:
        LDA     FILEFCB+0EH         ;FIXUP FCB FOR REWRITE
        ANI     07FH
        STA     FILEFCB+0EH
        XRA     A
        STA     FILEFCB+32
        LXI     D,BUFFER            ;SET DMA ADDRESS FOR WRITE
        MVI     C,STDMA
        CALL    BDOS
        LXI     D,FILEFCB           ;POINT FOR REWRITE
        MVI     C,WRITER
        CALL    BDOS
        RET
;
;
;ENTRY POINT TO PASS 125 BYTES OF PARAMETERS TO THE PL/I-80
;PROGRAM
;
;       ENTRY FROM PL/I-80 CALL WITH A POINTER
;
GETPARM:
        CALL    GETPNT              ;GET POINTER ADDRESS TO (BC)
        LXI     H,BUFFER+3          ;OFFSET INTO BUFFER
        LXI     D,125               ;SIZE OF PARAMETER AREA
GPAR1:
        MOV     A,M                 ;FROM BUFFER
        STAX    B                   ;TO PL/I DATA AREA
        INX     H                   ;BUMP POINTERS
        INX     B
        DCX     D                   ;DECREMENT SIZE COUNTER
        MOV     A,D                 ;CHECK IF DONE
        ORA     E
        JNZ     GPAR1
        RET                         ;BACK TO PL/I-80
;
;
;ENTRY POINT TO MOVE 125 BYTES OF PARAMETERS FROM PL/I-80
;PROGRAM TO LOCAL BUFFER
;
;       ENTRY FROM PL/I-80 CALL WITH A POINTER
;
PUTPARM:
        CALL    GETPNT              ;GET POINTER ADDRESS TO (BC)
        LXI     H,BUFFER+3          ;OFFSET INTO BUFFER
        LXI     D,125               ;SIZE OF PARAMETER AREA
PPAR1:
        LDAX    B                   ;FROM PL/I DATA AREA
        MOV     M,A                 ;TO BUFFER
        INX     H                   ;BUMP POINTERS
        INX     B
        DCX     D                   ;DECREMENT SIZE COUNTER
        MOV     A,D                 ;CHECK IF DONE
        ORA     E
        JNZ     PPAR1
        RET                         ;BACK TO PL/I-80
;
;
        END
;
;
;...+++END OF FILE
```

Listing 2

```
            /* * * * * * * * * * * * * * * * * * * * * *
             *                                           *
             *  ROUTINE TO DEMONSTRATE AND TEST THE      *
             *  FIRST SECTOR ACCESS ROUTINE              *
             *                                           *
             *    WRITTEN BY:                            *
             *            MICHAEL J KARAS                *
             *            MICRO RESOURCES                *
             *          2468 HANSEN COURT                *
             *        SIMI VALLEY, CA 93065              *
             *           (805) 527-7922                  *
             *                                           *
             * * * * * * * * * * * * * * * * * * * * * */
/*
      FIRST OBJECT CODE SECTOR OF AN EXECUTABLE IMAGE OF
      THIS PROGRAM MAY BE ACCESSED BY FOLLOWING THE CODE
      SEQUENCE SHOWN BELOW.

      THE DATA IN THE SECTOR MAY BE MODIFIED SIMPLY
      BY FOLLOWING THE SECOND SEQUENCE OF OPERATIONS
      SHOWN BELOW.                                        */


TESTPROG: PROC OPTIONS (MAIN);

   DCL
     PARM          CHAR(125) BASED (P),  /* FULL BLOCK DATA */
     DATA2 (5)     CHAR (25) BASED (P),  /* ARRAY FORMAT OVERLAY */

     1 DATA_ACCESS BASED (P),            /* ELEMENTAL OVERLAY */
       2 CURREC    BIN FIXED (15),       /* LAST ENTERED RECORD NO. */
       2 FDATE     CHAR (6),             /* LAST ACCESS DATE MMDDYY */
       2 FTIME     CHAR (6),             /* LAST ACCESS TIME HHMMSS */
       2 TOTAL1    DEC FIXED (6,2),      /* RUNNING TOTAL FIELD #1 */
       2 TOTAL2    DEC FIXED (6,2),      /* RUNNING TOTAL FIELD #2 */
       2 TOTAL3    DEC FIXED (6,2),      /* RUNNING TOTAL FIELD #3 */

     P             POINTER,

     RDBASE        ENTRY,                /* READ FIRST SECTOR ENTRY */
     WRBASE        ENTRY,                /* WRITE FIRST SECTOR ENTRY */
     GETPARM       ENTRY (POINTER),      /* GET PARAMETER DATA */
     PUTPARM       ENTRY (POINTER);      /* PUT PARAMETER DATA */


     ALLOCATE PARM SET (P);    /* ESTABLISH DATA STORAGE AREA */
     CALL RDBASE;              /* READ FIRST SECTOR INTO BUF */
     CALL GETPARM(P);          /* FILL LOCAL STRUCTURE WITH DATA */


/*************************************************************************
   put your parameter sector access program here. Use fetched
   information, generate new data or configuration parameters
   or what ever you please.
 ************************************************************************/


     CALL PUTPARM(P);          /* TRANSFER NEW DATA TO BUF */
     CALL WRBASE;              /* WRITE BACK ONTO DISK */

     END TESTPROG;

/*+++...END OF PL/I-80 TEST PROGRAM */
```

Assembler Programming Tutorial: Input/Output Instructions

Ward Christensen

In previous installments of the tutorial, I covered most of the instructions for doing data manipulation within the 8080. This month I'll show how the 8080 communicates with the outside world. The 8080 has two instructions for input and output.

IN and OUT

The IN instruction transfers 8 bits of data into the 8080 accumulator from some device. The OUT instruction transfers 8 bits of data from the accumulator, to some device.

The instruction allows accessing up to 256 devices. You code the instructions:

```
IN addr
or
OUT addr
```

where addr is a number from 0 to 255 (or hexadecimal 0 to FF).

The data transfer is always parallel, which means that all 8 bits of data are transferred at the same time. Therefore, special hardware is required for communication with devices which do not present all their data as 8 parallel bits. For example, if a Teletype(R) is attached to an 8080, the attachment must contain a device for serially getting the bytes from the keyboard, and then presenting them in parallel to the 8080. It must also contain a device for taking parallel bits from the 8080 and presenting them to the TTY in a serial fashion. This device is the UART - Universal Asynchronous Receiver/Transmitter, mentioned in the "terms" section of this tutorial.

One of the commonest devices attached to a hobbyist's 8080 system is a keyboard. I'll use this device, and show some examples of the programming necessary to get data from a keyboard. I will have to get into just a bit of hardware, however, to explain how a keyboard interface works.

A typical interface for a keyboard sends data on 2 addresses, for example 0 and 1. One address is for status information, and the other for data.

These are not memory addresses, but *are* addresses, which are placed on the address bus during the time of INP or OUT instructions.

The address which gives us status is frequently called the "status port". If you see or hear the word "port", you may consider it to be the "address" through which an input or output device is referenced.

When a key is pressed, the interface presents a single bit of information on the status port. I am not concerned with what data is on the other 7 bits, so I must isolate the single bit we are interested in. If that bit has the proper value (defined by the implementor of the interface, as being either 0 or 1), then that is an indication that a key has been pressed, and we must read the character from the data port.

In this example, I will assume that the status bit comes in at the rightmost bit of the status port:

x x x x x x x #

where "x" are bits I don't care about, and "#" is the bit showing a key has been pressed. Here's a sample program:

```
WAIT IN   0    ;INPUT STATUS
     ANI  1    ;GET READY BIT
     JNZ  WAIT;LOOP TIL READY
     IN   1    ;INPUT DATA
```

Looking at this program, we see that the hardware interface was implemented so that the status port "ready bit" is "1" (sometimes called "high") when no key has been pressed. Therefore, when I input the status port, (IN 0), and "AND" it with a 1, I am isolating just the single, rightmost bit of the status port. If this bit is a 1, then the Program Status bits will be set to not zero, because the accumulator contains a 1 after the ANI instruction.

Thus the JNZ instruction causes the IN STAT to be executed over and over, until the tested bit becomes 0. Control will then fall through the JNZ instruction, and will execute the IN DATA instruction, and read the data keyed.

If the status bit were somewhere other than the rightmost bit of the status port, we would have to "ANI" with the proper value to isolate the bit. For example, if the status bit were the second from the left, i.e.

x # x x x x x x

then we would have to ANI with 40 hex, because 40 hex is 01000000 in binary bits. The CP/M assembler accepts this in several forms, the most common of which are binary, and hexadecimal. In binary, it is coded as 01000000B, and in hex as 40H.

Let's turn to the data to be read from the keyboard. When working with character ASCII data, the highest bit is usually ignored, as this is the parity bit in ASCII. Therefore, you will frequently see input routines such as this coded with:

```
ANI 7FH ;TURN OFF PARITY
```

as the last instruction. This is so that instructions such as:

```
CPI 'R'
```

will test for the proper value. The 'R' gets stored in memory by the assembler as a 52 hex. If the keyboard we are dealing with sends in the high bit as "on" (i.e. "1"), then IN DATA places the value B2 hex in the accumulator, and the compare to 'R' (52 hex) wouldn't work, even though they both are the letter R. The ANI 7FH takes care of this.

Not all computers have INP and OUT instructions, yet they are perfectly able to perform the same functions as the 8080 with it's IN and OUT instructions. These computers use "memory mapped" input and output.

This means that the interface (such as to a keyboard) watches for a particular location in memory to be read, instead of watching for an input instruction. When it sees this, it puts the status or data information into the computer. This means that we cannot also have memory at the same address as we have our input or output device.

This is a rather minor drawback of memory mapped input and output, namely that it "steals" some of the memory addresses. Thus, in a typical microcomputer which is capable of addressing 64K (65,536) bytes of information, some block of memory is set aside for addressing input and output devices, therefore limiting our maximum memory to some value, such as 60K.

There is nothing which prevents the 8080 from using memory mapped input and output. Several of the manufacturers of plug-in boards for the popular S-100 computers use memory mapped I/O.

Now, let's tie in all the instructions which we have learned, and code a short program which inputs a LINE of data from a keyboard, echoing every character as it is typed, and stops when the carriage return is pressed. At this time, it echoes a linefeed, (so we don't have to bother pressing it after the carriage return). This routine will be coded as a subroutine, i.e. a routine which we can CALL. As an added convenience, the calling routine passes the address of the buffer to be filled, in the HL register:

```
LXI   H,BUFFER ;POINT TO
               ; BUFFER
CALL LINPUT    ;INPUT A LINE
```

I will use some abbreviations: C/R for carriage return, L/F for line feed. I have assembled this program at 100 (hex) so you can become familiar with how an assembly listing looks. (See the Listing which follows this article.)

Next month, I'll cover the few instructions that haven't been seen so far, and will begin showing lots of examples. Particularly, these examples will be useful in day-to-day CP/M programming.

```
                    ;-----------------------------------
                    ;       Sample program for 8080
                    ;   assembler programming tutorial
                    ;-----------------------------------
                    ;
                    ;Subroutine to input a line of
                    ;data from a TTY to a buffer
                    ;
                    ;The routine calling "linput" must
                    ;supply the buffer address in HL,
                    ;via an "LXI H" instruction
                    ;
                    ;The routine echoes all keyed characters
                    ;
                    ;A linefeed is automatically echoed after
                    ;the carriage return is typed
                    ;
                    ;The following instruction "origins" or
                    ;"starts" the assembly at 100 hex.
                    ;
0100                        ORG     100H
                    ;
0100 =              LINPUT  EQU     $       ;entry point
0100 CD1301         LOOP    CALL    KEYIN   ;get a character
0103 CD1F01                 CALL    TYPE    ;echo the char
                    ;
                    ;Note that if "type" doesn't save the
                    ;character, we would have to
                    ;
                    ;       push    psw
                    ;and
                    ;       pop     psw
                    ;
                    ;To keep the character for the C/R test
                    ;
0106 77                     MOV     M,A     ;store in memory
0107 23                     INX     H       ;point to next
                                            ;loc in buffer
0108 FE0D                   CPI     0DH     ;is it C/R?
010A C20001                 JNZ     LOOP    ;..no, loop
                    ;
                    ;We got a C/R, so echo the linefeed
                    ;and return
                    ;
010D 3E0A                   MVI     A,0AH   ;get L/F char
010F CD1F01                 CALL    TYPE    ;echo the L/F
0112 C9                     RET         ;return to caller
                    ;
                    ;Subroutine to get a character from
                    ;the keyboard
                    ;
0113 DB00           KEYIN   IN      0       ;input status
0115 E601                   ANI     1       ;isolate
0117 C21301                 JNZ     KEYIN   ;loop not ready
```

(continued next page)

Full ASCII Keyboard for Apple CP/M

Matthew Von-Maszewski

To the delight of Apple II owners and retailers, Microsoft created a Z80 card and configured a CP/M for the Apple. Microsoft's work opened up a whole new area of software to the Apple II. Unfortunately, the Apple II keyboard does not have enough keys or support logic to provide a full ASCII character set to CP/M programs that require one.

This problem was brought to my attention in two ways. First I tried to run MicroPro's WordStar and WordMaster that I copied from my S-100 CP/M computer to the Apple. The 24*80 video card in my Apple filtered some of my control characters, never letting CP/M see them. Also, due to Microsoft's keyboard redefinition program, some of the characters that did not get filtered were changed. I had found one way around this problem when Steve Jenkins, manager of a Dallas computer retail store, restated the problem in a different way. He pointed out that Videx produced a board for approximately $125.00 that did what my easy patch did and more. Steve challenged me to copy the Videx capabilities through software that he could give away with Apples he sold. This article details the comprehensive patch that solves the above problems.

The patch accomplishes the following functions to allow any UNMODIFIED CP/M program to run on the Apple:

1. All ASCII characters are made available at the keyboard. Some require the pressing of the control key and another key, but they are all available.
2. All CONTROL characters remain intact. The extra ASCII characters such as {, [, },], rub and delete are put on number keys and symbol keys that normally have no CONTROL functions.
3. A software uppercase lock is turned on and off through use of Control 0 (Zero).
4. Characters are read directly from the keyboard - not through the video interface - to prevent any characters from being filtered out.

Note: This patch only worked in CP/M for full keyboard, but it does act as the standard SHIFT key modification for Apple-based software.

This patch eliminates the need for a special Apple WordStar, or other software modified for the Apple. In the case of Apple WordStar, it simplifies the elaborate key combinations that MicroPro had to develop to enable WordStar to run on the Apple, and it opens the door for the Apple to use other software (like WordMaster) which has no special Apple version to compensate for the limited keyboard.

The software patch requires the support of two wires in order to perform. The first wire is the standard SHIFT key modification that has been used on the Apple for several years. The second wire is a modification to the CONTROL key in the same manner as the SHIFT key. This hardware costs less than a dime, a sizable savings when compared to $125.00!

The wires are added to the Apple in one of two ways, depending on which revision keyboard your Apple has. All of the later revisions have a second PC board below the keyboard. Earlier revisions do not have this second board at all. One of the features of the second board is a RESET protect switch that allows the Apple to be reset only when both the RESET key and the CONTROL key are pressed together. If your Apple does not have this built-in feature, you have the earlier revision keyboard.

To modify the later revision boards:

1. Orient the computer with the keyboard towards you and the cover open.
2. Locate the 25 pins that connect the keyboard to the second board attached to it (not the big mother

Assembler Programming Tutorial (continued from previous page)

```
011A DB01              IN      1         ;read data
011C E67F              ANI     7FH       ;delete parity
011E C9                RET
                ;
                ;Subroutine to type a character.
                ;this routine preserves the char
                ;in a
                ;
011F F5         TYPE   PUSH    PSW       ;save the char
                ;
                ;Loop, waiting for character ready
                ;
0120 DB00       TWAIT  IN      0         ;get status
                ;
                ;Note that the bit we isolate (80H)
                ;is different when testing to see if
                ;the output is ready:
                ;
0122 E680              ANI     80H       ;isolate
0124 C22001            JNZ     TWAIT     ;loop not ready
                ;
                ;The TTY is ready to accept the char.
                ;
0127 F1                POP     PSW       ;get the char
0128 D301              OUT     1         ;output it
012A C9                RET               ;and return
```

```
;       PATCH.ASM          09/05/81 23:32
;
;Written by:
;       Matthew Von-Maszewski,
;       Staley Computer Associates
;       P.O. Box 5189
;       College Station, TX  77840
;
;
;       CP/M keyboard patch to allow control characters
;               to get operating program without being
;               filtered out by the 24*80 card.
;
;       Note:  those using a M&R Supe'R'Term need the
;               2.5 version of the ROM on the Card.
;               This ROM can be attained from M&R
;               or the author of this program.
;
;       There should be no characters redefined under CONFIGIO.
;               We give you all of them.  Beat that one
;               Micro Soft.
;
;       See accompaning text for hardware modification to
;               to support this routine.
;

PATCH   EQU     0F280H          ;LOCATION OF PATCH
                        ;placed in punch because tty area has
                        ;stuff in it no one tells us about.
                        ;       gee thanks Micro Soft.
FALSE   EQU     0
TRUE    EQU     NOT FALSE

SHIFT   EQU     TRUE            ;KEYBOARD HAS MOD FOR SHIFT KEY ON SWITCH 2
SHLOCK  EQU     TRUE            ;KEYBOARD HAS MOD FOR SHIFT LOCK ON SWITCH 1

        ORG     100H
        DB      1               ;NO OF PATCHES
        DW      PATCH           ;DESTINATION OF PATCH
        DW      LC2-LC1         ;LENGTH OF CODE
        DB      1               ;PATCH TYPE 1
        DB      2               ;PATCH VECTOR 2
        DW      PATCH           ;WHERE TO SEND VECTOR

LC1     EQU     $
BIAS    EQU     PATCH-$
L1:     LDA     0E000H          ;GET VALUE OF KEYBOARD
        CPI     80H             ;CHECK FOR HIGH BIT SET
        JC      L1+BIAS         ;GO BACK IF NO KEY
        ANI     7FH             ;STRIP OFF HIGH BIT
        STA     0E010H          ;TURN OFF KEYBOARD STROBE

        IF      SHIFT

        if      shlock
        push    b               ;save this reg till later
        mov     b,a             ;save the char
        lda     flag+bias       ;see if under shift lock
        inr     a               ;test flag
        mov     a,b             ;get character back
        jz      12+bias         ;jump is shift lock
        CPI     41H             ;SEE IF CHAR IN LETTER AREA
        jc      12+bias         ;leave this section if not alpha
        endif

        if      not shlock
        cpi     41H             ;see if char in letter area
        jc      12+bias+1       ;leave this section if not alpha
        PUSH    B               ;SAVE THIS REG.
        endif

        MOV     B,A             ;PUT CHAR INTO B FOR SAFE KEEPING
        LDA     0E063H          ;GET PUSH BUTTON
        ANI     80H             ;STRIP OFF OTHER STUFF
```

board that extends through out the Apple case). Pin 1 is the left-most pin (closest pin to the side of the Apple with the power supply).

3. SHIFT key wire-Attach one end of a wire to the second pin from the right (pin 24). Run the other end of the wire to the game I/O connector (where you plug in the paddles).
4. CONTROL key wire-Attach one end of a second wire to the third pin from the left (pin 3). Run the other end of this wire to the game I/O connector also.

To modify earlier revision boards:

1. The bottom plate of the Apple must be removed and separated from the shell to expose the underside of the keyboard. Orient the shell with the keys down and toward you.
2. The two wires must be soldered onto the board as shown in the illustration. The wires must be long enough to reach the game I/O connector when the Apple is reassembled.
3. Reassemble the Apple.

For both revisions the two wires must be attached to the game I/O connector. This can be done in two easy ways. First, the bare ends of the wires may be inserted into the socket holes. Second, the wires may be soldered onto a sixteen-pin dip header; the dip header is then inserted into the socket.

The SHIFT key wire must be connected to pin 4 and the CONTROL key wire must be connected to pin 3.

The software part of this patch is added to the CP/M system by way of Microsoft's CONFIGIO program that comes with the Apple CP/M. After you have entered the PATCH.ASM as listed, use the CP/M assembler to produce a hex file. Next the hex file must be converted to a COM file. Finally the patch is moved into the CP/M. The detailed instructions follow:

1. Enter the PATCH.ASM using ED.
2. Assemble the patch using the following command
   ASM PATCH.AAZ
3. If you have any errors during assembly, edit them, and return to step 2.
4. Translate PATCH.HEX to PATCH.COM (note: this is not an executable COM file) using this command:

(continued next page)

LOAD PATCH

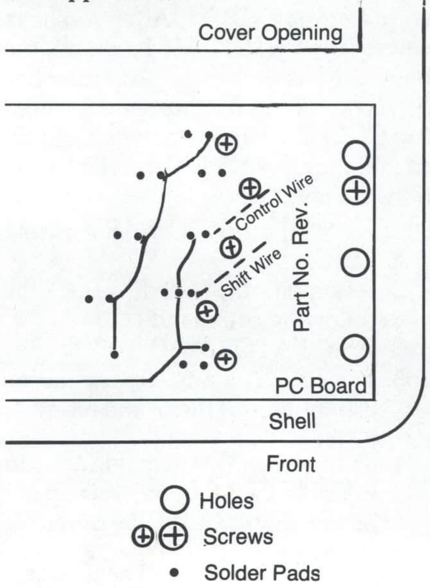
5. Execute the configure program: MBASIC CONFIGIO
6. Go to section 2 of CONFIGIO, keyboard redefinition, and delete any redefinitions present. See the Microsoft manuals (pp. 5-16, Volume II in my manuals) for details on CONFIGIO.
7. Go to section 3 of CONFIGIO, load user patch. Read in the patch, PATCH.COM. Again, see the Microsoft manuals for how CONFIGIO is operated.
8. Go to section 4 of CONFIGIO, read/write system, and write the current system with the patch out to disk.

The patch is now present on the disk and is active in current memory, so you can immediately begin trying it out. See the end of PATCH.ASM for the table of what control keys produce what.

In closing, this patch is available for public use, but I reserve the right of its use for sale. Should you wish to sell it or provide it with computers you sell, I reserve the right to give permission for such use and to request proper compensation.

Author's Note: If you do not wish to type in the patch yourself, you can send me $15.00 and I will send you a disk with the software prepared to be added to the system through CONFIGIO.

If you do not wish to modify your Apple yourself, there are several options available. First, try taking this article, along with your Apple, to your local computer store; ask them to do the modification. If this is not possible, try to find someone capable of modifying the Apple for you.



```
         RRC                          ;MOVE TO BIT 6
         RRC                          ;MOVE TO BIT 5
         ORA      B                   ;DO THE STUFF
L2:      POP      B
         cpi      40h                 ;check for wrong keys in
         jz       14+bias             ;make @ a capital P like it should be
         cpi      5eh                 ;see if ^
         jz       13+bias             ;make ^ a capital N
         cpi      5dh                 ;see if ]
         jnz      15+bias             ;nope: leave rest of characters alone
13:      ani      4fh                 ;fix for both M & N
         jmp      15+bias             ;kitty fixed
14:      mvi      a,50h               ;now a P
15:

         ENDIF

         IF       SHLOCK
         push     psw                 ;save character
         LDA      0E062H              ;GET PUSH BUTTON
         RAL                          ;
         jc       18+bias             ;skip rest if control key is up
         pop      psw                 ;get the character

         cpi      2dh                 ;check lower bound
         jc       18+bias+1           ;below what we are looking for
         cpi      3bh                 ;check upper bound
         jnc      18+bias+1           ;above what we are looking for
         cpi      30h                 ;see if shift lock
         jz       17+bias             ;it is. jump

         ;translate area

         push     h                   ;save
         lxi      h,table+bias        ;get base of translate table
         sui      2dh                 ;strip off unneeded ASCII
         add      1                   ;do offset
         mov      l,a                 ;save offset
;        jnc      16+bias             ;see if a carry into h
;        inr      h                   ;add 1
16:      mov      a,m                 ;get replacement value
         pop      h                   ;restore hl
         jmp      18+bias+1           ;finally finished

17:      lda      flag+bias           ;
         cma                          ;change flag
         sta      flag+bias           ;put flag back in new state
         jmp      11+bias             ;go back for next char
flag     db       0                   ;flag starts off (lower case)

table:            ;character translate table
         db       5fh                 ;translate ctrl-= to rub (underline)
         db       7fh                 ;ctrl-. to delete
         db       5ch                 ;ctrl-/ to \
         db       0                   ;shift lock gets intercepted
         db       7ch                 ;ctrl-1 to |
         db       00h                 ;ctrl-2 to Null
         db       7eh                 ;ctrl-3 to ~
         db       7bh                 ;ctrl-4 to {
         db       7dh                 ;ctrl-5 to }
         db       5eh                 ;ctrl-6 to ^
         db       60h                 ;ctrl-7 to `
         db       5bh                 ;ctrl-8 to [
         db       5dh                 ;ctrl-9 to ]
         db       40h                 ;ctrl-: to @

18:      pop      psw                 ;get char back

         endif

         RET
LC2      EQU      $
```

dBASE-II, Version 2.3

Reported by Michael Olfe

This version adds new features, extends many of the previous features, and fixes some bugs in version 2.02. The integral editor makes it possible to create and modify "CMD" files from within "DBASE". In addition, several (as yet) undocumented features are supported. "DBASE.COM" is now 18K compared to 16k for the previous version.

1. The install program now allows up to 11 characters for cursor addressing and more characters for highlighting, allowing a wider range of terminals to be supported. The standard installation menu includes Osborne, Apple, Adds Viewpoint, Xerox 820, and HP2621.

2. There is a new command, BROWSE, which displays data from up to 19 records on the screen, allows editing and scrolling of the data.

3. New functions

FILE(<string exp>): evaluates to .TRUE. if file named **<string exp>** is on the disk.

TRIM(<cstring>): removes trailing blanks from <cstring>

TYPE(<exp>): gives type of <exp> - character, numerical, or logical

4. Extensions

@ <exp>,<exp> say <string> now allows <exp> to be a numeric variable, unlike the previous version.

! is now allowed as a format character in GETs, converting lower to upper case.

CLEAR GETS clears all pending GETs and leaves screen intact.

DO CASE, CASE <exp>, OTHERWISE, ENDCASE allow Pascal-like case statements instead of nested if-then-else constructs.

MODIFY COMMAND allows creation and editing of "CMD" or any other text files.

PACK now automatically adjusts the index file.

REPORT [FORM <formfile>] PLAIN causes page numbers and date at the top of a report to be suppressed.

USE <database> INDEX <indexfile>[,<indexfile>...] now allows up to 7 index files to be used with each database.

5. **SET** has many new parameters. The following "SETS" can be On or OFF:

SET LINKAGE ON makes all sequential commands (LIST,REPORT,SUM, etc.) perform positioning on both Primary and Secondary Databases.

SET COLON ON bounds GET data items with colons.

SET BELL ON makes bell ring on error condition.

SET ESCAPE ON allows escape character to abort.

SET EXACT ON requires complete match on string compares.

SET INTENSITY ON Dual intensity used in full-screen operations.

SET DEBUG ON makes output from ECHO and STEP go to printer instead of console.

SET CARRY ON carries over data from previous record when APPENDing records in full-screen mode.

SET CONFIRM ON suppresses a skip-to-next-field until a control key is typed.

SET EJECT ON causes REPORT command to eject a page before beginning report.

SET RAW ON places spaces between fields when DISPLAY and LIST are used without fields list.

The following SET commands take strings or numbers as parameters:

SET DATE TO mm/dd/yy sets system date.

SET INDEX TO <index file> [,<index file>, ...] identifies up to 7 index files to be used for future operations.

SET MARGIN to N allows user to control left margin in a report.

6. Undocumented additions:

PEEK, Poke Read and write to any location in memory

CALL <memvar>

SET CALL to <memvar>

7. Error trapping

Judging from the large number of error messages listed in the manual, there seems to have been a significant increase in the amount of error trapping.

8. Changes to manual

The new manual includes tables of commands, organized alphabetically and by function, a list of error messages, and new examples and descriptions.

WordStar Modifications for the Epson MX-100 Printer

Bob Kowitt

The Epson line of printers lend themselves very well to word processing with Micropro's WordStar if various of the printers' functions are programmed into the WordStar drivers.

The USER4 listing provided in the manual indicates the program locations to be changed. In addition to altering the running program, the on-screen print menu should be altered, to reflect changes in the operation of WordStar.

I modified WordStar to run on my MX-100 printer. The control codes for the MX-80 are different and a form of my modifications may also be adaptable for other dot-matrix printers. Substitute the control codes for your printer.

The changes that I wanted were :

Print enlarged characters

```
        WORDSTAR
    MODIFICATIONS
```

Print condensed characters

```
        A list of the modifications.
```

Print boldface

Print eight lines per inch

```
        This a block of copy
        that is being printed
        at eight lines per
        inch.
```

Print six lines per inch

```
        And another block of
        copy that is being
        printed at the de-
        fault of six lines
        per inch.
```

Turn off the paper end detector to permit single sheets.

The print option menu has two commands that were designed to change printer characteristics from ten characters per inch to twelve characters per inch. Since my printer does not have a way to switch between these two pitches, I decided to change these Wordstar commands to allow a switch between the standard width character and the enlarged character produced by the Epson MX-100.

I wanted to be able to use toggle controls wherever possible. The ribbon color change was one I could modify since I had no ribbon color change ability on the Epson. This was an ideal place to have a toggle between condensed mode and normal. I realize there is an inconsistency in having a toggle for condensed mode and two separate controls for the wide mode. I settled for this inconsistency to permit using as few of the controls as possible.

The other toggle commands on the left side of the menu involve the reprinting of a line for underscore, strikeout, superscript and subscript. Since the Epson does not backspace, these commands cause a complete reprint of the line, so were not available to me for my purpose.

It would have been preferable to have been able to use the ↑P↑B toggle provided by WordStar for the boldface (called emphasized by Epson) but the Epson needs two separate signals for emphasized-on and emphasized-off.

Since I did not want to sacrifice any more of the standard WordStar printer controls, I decided to use USER1, USER3 and USER4 controls provided by the print menu. USER3 is used for forcing 8 lines per inch and USER4 for returning to 6 lines per inch. USER1 is used for deselecting the paper end detector to allow the printer to print on single sheets properly. Since I wanted, under normal circumstances to have the paper end detector operable and the printer in the 6 lines per inch mode, the WordStar end-of-print closing functions must be modified to restore these modes.

To sum up the changes to be made:

1 - Alternate character width	→ expanded	↑P↑A
2 - Standard character width	→ normalize from wide	↑P↑N
3 - Ribbon color change	→ condensed (toggle)	↑P↑Y
4 - User (1)	→ deselect paper end detector	↑P↑Q
5 - User (3)	→ force 8 lines/inch	↑P↑R
6 - User (4)	→ force 6 lines/inch	↑P↑E

The last three are set to normal upon ending the printing.

The first thing to be done is change the printer driver. After this is proved to be working, the messages must be changed to permit menu prompting. Use a copy from your master Word Star disc that is *already* modified for your terminal.

Place a copy of DDT or SID on the B:drive and type:

```
B:DDT WS.COM

S690 (cr)
```

DDT will respond (and you enter the boldfaced values):

```
690   0   FF     ;to send CR w/o LF for overprint
691   0   04     ;set to 4 strokes for bold
692   0   02     ;set to 2 strokes for double strike

S6B5 (cr)
```

```
6B5  0  01        ;1 character for expanded set
6B6  0  0E        ;set expanded

                  S6BA (cr)

6BA  0  01        ;1 character to reset expanded
6BB  0  14        ;reset expanded

                  S6C9

6C9  0  02        ;2 characters for paper end
6CA  0  1B        ;this is one
6CB  0  38        ; and the second

                  S6D3

6D3  0  02        ;2 characters for 8 lines/in.
6D4  0  1B        ;this is one
6D5  0  32        ; and the second

                  S6D8

6D8  0  02        ;2 characters for 6 lines/in.
6D9  0  1B        ;this is one
6DA  0  30        ; and the second

                  S6DD

6DD  0  01        ;1 character for toggle
6DE  0  0F        ;condensed type set

                  S6E2

6E2  0  01        ;1 character for toggle
6E3  0  12        ;condensed type off
```

Printer conclusion settings:

```
                  S6F8

6F8  0  04        ;there will be 4 characters
6F9  0  1B        ;lead in for paper end
6FA  0  39        ; selector on again
6FB  0  1B        ;lead in for 6 lines/in.
6FC  0  32        ; set on again
```

While I was at it, I decided to change the strikeout character from a '-' to a '/'.

```
                  S70B

70B  2D  2F       ;changes - to / strikeout
```

Change the message that comes up on invoking WordStar by using

```
DDT WS.COM
     D1B0
```

Modify the file as in Figure 1 to display the Epson MX-100 sign on message. This completes the changes to WS.COM and it must be saved to the disc:

```
SAVE 62 WS.COM
```

In order to change the print menu to reflect the changes made, you must modify the WSMSGS.OVR overlay. If you have the WordStar customization notes, you will find the appropriate areas and modify them according to instructions. Since I did not have them, I modified the menu using DDT. The area in WSMSGS.OVR that must be changed will be different depending upon the version of WordStar you are using. The following locations are for WordStar 3.0. If you are using a different version, search for the messages relating to the changes to be made and substitute in the same manner.

It was necessary to make room in the print menu for the new commands. The display of the other menus available is not usually required and, if it is, can be checked elsewhere. All the new commands will be prompted in this area.

The modification I used in Figure 3 is straightforward but I want to call your attention to the modifications shown at location 16D0H.

In WordStar 3.0, sections of the menus that are displayed in either enhanced, reverse or altered screen in some manner. These characters are filed as normal ASCII in WSMSGS.OVR and before sending them to the screen, the high bit is complemented to produce the alteration. Therefore, to display a message in normal mode in this area, one must have in the message file, a message with the high bit set. When displayed, this is complemented to provide normal display. For this reason, the sections of the menu that display in normal mode must be changed with DDT, by finding the message hidden in the file as a byte with the high bit set. That is, if we want to find "A", we cannot look for 41H but must find C1H. To find a space, normally 20H, we look for A0H. (Make the alterations shown in Figure 3.)

Now translate the changes at 16D0H and you will find they spell out:

```
———————————Printing Changes———————————
```

I also wanted to display the fact that USER1, USER3 and USER4 of the options were no longer available, so I put them in normal mode, as well, to contrast with the reverse display my terminal uses. This line is changed at locations 18BFH thru 18D1H inclusive. (See Figure 4.)

Save the new message file:

```
SAVE 112 WSMSGS.OVR
```

I hope you enjoy the result of these modifications. I know I have. If you can think of any more, please let me know.

I want to put out a request for information about a method of changing the 'dot' commands in WordStar. I would like to be able to modify the 'lines per inch' with the .LH command. The Epson MX-100 will allow line advance control in 72nd's of an inch. If I could do that, I would not need two of the controls used in this modification and could do something else with them.

I can be reached in care of *Lifelines*: Bob Kowitt, c/o *Lifelines/The Software Magazine*, 1651 Third Ave., New York, N.Y. 10028.

See next page for Figures.

Figure 1

```
#IB:WS.COM
#R
NEXT  PC  END
7100 0100 AFFF
#D1B0,1D5
01B0: 20 0F 00 20 45 70 73 6F 6E 20 4D 58 2D 31 30 30  .. Epson MX-100
01C0: 20 77 2F 45 6E 68 61 6E 63 65 64 20 54 79 70 65  w/Enhanced Type
01D0: 20 20 20 20 20 0F        .
```

Figure 2

```
#IB:WSMSGS.OVR
#R
NEXT  PC  END
7100 0100 AFFF
#I
#D16D0,1716
16D0: F3 A0 AD AD AD AD AD AD AD A0 20 20 20 7C 20 AD ..........   ! .
16E0: AD AD AD AD AD AD AD AD D0 F2 E9 EE F4 E9 EE E7 ................
16F0: A0 C3 E8 E1 EE E7 E5 F3 AD AD AD AD AD AD AD AD ................
1700: AD 20 20 20 20 20 0E 20 28 62 65 67 69 6E 20 61 .     . (begin a
1710: 6E 64 20 65 6E 64 29 nd end)
```

Figure 3

```
#D1720,17A0
1720: 74 69 6D 65 20 65 61 63 68 29 20 20 20 7C 20 41 time each)    ! A
1730: 7C 57 69 64 65 20 54 79 70 65 20 20 20 20 20 20 !Wide Type
1740: 20 20 20 7C 20 51 20 4E 6F 20 50 61 67 65 20 45    ! Q No Page E
1750: 6E 64 20 20 20 20 0E 20 42 20 42 6F 6C 64 20 44 nd     . B Bold D
1760: 20 44 6F 75 62 6C 65 20 7C 20 48 20 4F 76 65 72  Double ! H Over
1770: 70 72 69 6E 74 20 63 68 61 72 20 20 20 7C 20 4E print char   ! N
1780: 7C 4E 6F 72 6D 61 6C 20 54 79 70 65 20 20 20 20 !Normal Type
1790: 20 20 20 7C 20 52 20 38 20 4C 69 6E 65 73 2F 49    ! R 8 Lines/I
17A0: 6E n

#D17E0,17F6
17E0: 20 20 20 7C 20 45 20 36 20 4C 69 6E 65 73 2F 49    ! E 6 Lines/I
17F0: 6E 2E 20 20 20 20 0E n.      .

#D1810,1847
1810: 74 6F 6D 20 73 70 61 63 65 20 20 20 20 7C 20 59 tom space     ! Y
1820: 20 43 6F 6E 64 2E 54 79 70 65 20 54 6F 67 67 6C  Cond.Type Toggl
1830: 65 20 20 7C 20 20 20 20 20 20 20 20 20 20 20 20 e   !
1840: 20 20 20 20 20 20 0E 20        .
```

Figure 4

```
#D18B0,18E6
18B0: 65 72 70 72 69 6E 74 20 6C 69 6E 65 20 7C 20 D1 erprint line ! .
18C0: DB B1 DD 20 57 28 32 29 20 C5 A8 B3 A9 20 D2 DB ... W(2) .... ..
18D0: B4 DD 20 20 20 20 20 20 20 20 20 20 20 20 20 20 ..
18E0: 20 20 20 20 20 20 00        .
```

A Review of T/MAKER II — Part 3
Tablemaking with T/MAKER

Raymond Sonoff

This article describes some of the features of T/MAKER II that might be used to create tables. Recall that T/MAKER stands for Table MAKER and that text editing operations can be included as part of the data calculation capabilities of this software product. (See previous reviews in *Lifelines* January 1982/Vol. II, No. 8 and February 1982/Vol. II, No. 9 issues for additional details.) Examples of table generation involving engineering and business applications are given. Observations, assessments, and some personal opinions regarding the applicability of T/MAKER II to various tasks attempt to close out this review series.

How T/MAKER accomplishes its operations according to associated symbols will be illustrated by presenting three Figures for each example: a) the initially CREATEd Table; b) the COMPUTEd Table; and c) the CLEANed final Table. These will be labeled TABLENO#.RAW, TABLENO#.CPT, and TABLENO#.CLN, respectively.

T/MAKER command sequences for these respective files to be produced and stored would be : CREATE TABLENO#.RAW EDIT SAVE; RENAME TABLENO#.CPT COMPUTE SAVE; and RENAME TABLENO#.CLN CLEAN SAVE. Once the initial table is generated and stored, the COMPUTE function would be invoked. The first operation that is done automatically by this function is alignment of each column's raw data decimal points according to the EXample line which must always precede data line computations. The positions of the "9's" fields defines the width of each column, and only numbers and symbols that lie within those column positions will be processed during a COMPUTE operation. Note that associated formatting, equations, and symbols remain as part of the SAVEd TABLENO#.RAW and TABLENO#.CPT files, whereas invoking of the CLEAN function as an operation to be performed on the COMPUTEd file, namely TABLENO#.CPT, results in a stripping away of the EXample line, Zero Values (ZV) line, row and column equations, and any other format information. Once TABLENO#.CLN is completed, the PRINT command will provide a hardcopy printout if it is invoked. If there is no need for storing this file, the SAVE command can be omitted. These three stages of table generation correspond to Figures with suffixes of (a), (b), and (c), respectively.

Engineering

The presence of expressions involving exp(at) where a is a constant and t is the parameter of time frequently appear in engineering computations. Figures 1 (a),1 (b),and 1 (c) provide illustrations of how T/MAKER can be utilized to perform such calculations and place the values in a convenient tabular format.

Business

Both small and large business enterprises desire to have sales projections information readily at hand. How T/MAKER can be used to provide sales projections calculations is clearly illustrated in Figure 2 (c).

The first question you will probably ask when seeing Figure 2 (c) is, "How did all those numbers get generated, and what steps had to be performed to reach that point?" By first examining the initial TABLENO2.RAW file of Figure 2 (a), you will quickly become appreciative of the various associated formatting and equation codes that result in generation of Figures 2 (b) and 2 (c). Figure 2 (b) is discussed as an example of COMPUTE on one panel of the Quick Reference Card (QRC) that accompanies the T/MAKER II 3-ring binder manual. This example provides an excellent demonstration of how much analysis can be performed on even a minimal amount of data.

The number string 1234567 shown preceding Figure 1 (a) is present so that you will recognize that this region is normally reserved for basic equation symbols and column equation codes. Whenever text precedes the actual Table's EXample line, however, you will have no problems even if you desire to use any of the first seven column positions for text. Therefore, TABLENO1.TXT can be "text" that is an integral part of the overall file TABLENO1.RAW during its initial creation. When text material is entered into any of the first seven column positions *below* the EXample line, that information would be deleted when and if a CLEAN operation occurred.

Final Comments

I have found T/MAKER II to be a wonderful software tool. It is forgiving in its calculations; it lets you know where you are in a "sea" of rows and columns via INFO (or ESC ? when in EDIT mode) commands, and you can keep only the files you desire, along with any supporting text you wish to include.

Having spent a considerable amount of time with this product, what do I recommend you do if you should decide to purchase T/MAKER II? Foremost of all, spend the time reading every single page of the manual, working every single example that is there. I made some rather simple errors during my indoctrination period, and these would have been minimized if I had simply plodded through every example provided in the manual. Second, experiment! Try various programming arrangements to see what seems to work best for you. I have plenty to learn still!

T/MAKER II has shortcomings as of this review time. This includes inability to present information in exponential notation, the need to essentially become intimately familiar with a Reverse Polish Notation-like calculation system: there are no parentheses involved, as there are for algebraic operating systems. Finally, you will have

(continued next page)

to get used to some cumbersome keystroke sequences (unless you want to redefine some of them).

Overall, I would rate this product as an excellent "assistant" for all types of report generation and calculations. Of course, if you need do only a simple calculation, such as could be done on a four-function calculator, don't bother with setting up the T/MAKER. However, if you will be doing rather frequent data manipulations, forecasting, or engineering calculations, it sure beats writing programs in BASIC. Good luck!

(Editor's Note: See the New Version report on Version 2.51 of T/MAKER II, released just as *Lifelines* went to press.)

TABLENO1.TXT

```
1234567(blank leading spaces)
       Deflection of an electron beam in an oscilloscope is
       expressed by the equation for sweep.  For a particular CRT,
       the sweep equation is given by:

                                           6
             D (t)  = 15.6 exp (-0.342 x 10  t)

       where D  is a function of time and is expressed in inches
       and   t  is expressed in microseconds.

       Determine the electron beam deflection for times of  0, 0.20,
       0.40, 0.62, 0.86, and 1.13 microseconds.


                             TABLENO1.RAW

       Time, t,     (at)        exp(at)  [1 - exp(at)]  Deflection,
       microsec. (a=-0.342)                                 in.
1234567
ex     9999.99   9,999.999     99.999       99.999        99.999
zv        zero          .0      0.           0.0           nil

ac1         +           =
ac2*               -0.342
 ac3                  +          exp
ac4                                +          sfo
ac5                                          +             =
ac6*                                                     15.6
+         0
 +          0.20
  +     0.40
   +      0.62
    +        0.86
     + 1.13
```

Figure 1(a)

TABLENO1.CPT

```
       Time, t,     (at)       exp(at)  [1 - exp(at)]  Deflection,
       microsec. (a=-0.342)                                 in.
1234567
ex     9999.99   9,999.999     99.999       99.999        99.999
zv        zero          .0      0.           0.0           nil

ac1         +           =
ac2*               -0.342
ac3                   +          exp
ac4                                +          sfo
ac5                                          +             =
ac6*                                                     15.6
+         zero         .0       1.000        0.0           nil
 +        0.20     -0.068       0.934        0.066         1.031
  +       0.40     -0.137       0.872        0.128         1.995
   +      0.62     -0.212       0.809        0.191         2.981
    +     0.86     -0.294       0.745        0.255         3.975
     +    1.13     -0.386       0.679        0.321         5.000
```

Figure 1(b)

TABLENO1.CLN

```
       Time, t,     (at)       exp(at)  [1 - exp(at)]  Deflection,
       microsec. (a=-0.342)                                 in.


         zero          .0       1.000         0.0          nil
         0.20      -0.068       0.934         0.066        1.031
         0.40      -0.137       0.872         0.128        1.995
         0.62      -0.212       0.809         0.191        2.981
         0.86      -0.294       0.745         0.255        3.975
         1.13      -0.386       0.679         0.321        5.000
```

Figure 1(c)

TABLENO2.RAW

```
                          Sales Projections
_____________________________________________________________________________
                  1978      1979      1980      Total     Growth    1981      1985
                                                (000)      Rate
_____________________________________________________________________________
ex              999,999   999,999   999,999   9,999.9      99.9  999,999   999,999
zv                 -         -         -         -           -       -         -
ac1                +         +         +         =           -       -         -
ac2/                                           1000
ac3                -         *         +                   grw
uc4                                    +                   prj      +        ++++
uc5                                                                rnd       rnd

++      Store A  12,345    13,679    17,567
+ +     Store B  21,345    20,145    19,134
        _____________________________________________________________________
=//     Total
        _____________________________________________________________________

ex              9999.99   9999.99   9999.99     ,,999.9      99.9  9999.99   9999.99
+%      Pct.Store
+ %     Pct.Store
        _____________________________________________________________________
=       Total
        _____________________________________________________________________
```

Figure 2(a)

TABLENO2.CPT

```
                          Sales Projections
_____________________________________________________________________________
                  1978      1979      1980      Total     Growth    1981      1985
                                                (000)      Rate
_____________________________________________________________________________
ex              999,999   999,999   999,999   9,999.9      99.9  999,999   999,999
zv                 -         -         -         -           -       -         -
ac1                +         +         +         =           -       -         -
ac2/                                           1000
ac3                -         *         +                   grw
uc4                                    +                   prj      +        ++++
uc5                                                                rnd       rnd

++      Store A  12,345    13,679    17,567      43.6      19.3   20,956    42,434
+ +     Store B  21,345    20,145    19,134      60.6      -5.3   18,116    14,557
        _____________________________________________________________________
=//     Total    33,690    33,824    36,701     104.2       4.4   39,072    56,991
        _____________________________________________________________________

ex              9999.99   9999.99   9999.99     ,,999.9      99.9  9999.99   9999.99
+%      Pct.Store 36.64     40.44     47.87                14.3    53.63     74.46
+ %     Pct.Store 63.36     59.56     52.13                -9.3    46.37     25.54
        _____________________________________________________________________
=       Total    100.00    100.00    100.00                  -    100.00    100.00
        _____________________________________________________________________
```

Figure 2(b)

TABLENO2.CLN

```
                       Sales Projections
____________________________________________________________________
            1978      1979      1980      Total     Growth    1981      1985
                                          (000)      Rate
____________________________________________________________________

Store A    12,345    13,679    17,567      43.6      19.3    20,956    42,434
Store B    21,345    20,145    19,134      60.6      -5.3    18,116    14,557
____________________________________________________________________
Total      33,690    33,824    36,701     104.2       4.4    39,072    56,991
____________________________________________________________________
Pct.Store   36.64     40.44     47.87                14.3     53.63     74.46
Pct.Store   63.36     59.56     52.13                -9.3     46.37     25.54
____________________________________________________________________
Total      100.00    100.00    100.00                        100.00    100.00
____________________________________________________________________
```

Figure 2(c)

BASIC Business Functions

James E. Korenthal

This article presents a rationale for making extensive use of Microsoft BASIC's powerful DEF FN facility in certain business programming situations, and gives some examples of functions that I've recently employed to great advantage. Sorry if I've misled you... but you've got to admit, "BASIC Business Functions' is gonna look a helluva lot better on my resume than "Hacker's Guide to DEF FN capabilities.' You might say I'm "entitled.' (...bored already? Thinking of turning the page? Too bad – I guess you're not interested in winning a ZX-80... Read on...)

Okay, down to business. First of all, why are we using BASIC, anyway? Because it's the most highly structured, elegant language around? Because it's faster than a speeding bullet? If either of these reasons apply to you, skip to the next article, entitled "IBM 1620 Fundamentals."

The most common reason for using BASIC (assuming, of course, that it's not the only language you have on your micro) is BASIC's interpretive nature, and the inclusion of an "environment" in most BASIC implementations. This means that a consultant can sit down at his terminal and create, edit, monitor, debug, and finalize a well-documented program without ever leaving BASIC. I'm assuming, of course, that he's using a BASIC interpreter like MBASIC, not a compiler like BASCOM.

Now – if we're talking about an interpreter, what does this say about execution speed? Simply that we don't care very much about it! Sure, we'll write our program in such a way that the user doesn't sit for hours staring at a blinking cursor, but when push comes to shove, a few extra seconds of execution time don't mean diddly. This is especially true in business programming, where most functions are I/O bound (meaning that most of the computer's time is spent fiddling with peripherals, rather than doing actual number crunching).

A brief digression is in order here, about a common programmer's disease called "one-liner syndrome." Here's how this malady works: you spend a considerable amount of time developing a single line of code which does something really impressive. You then challenge all your friends to do the same thing in the smallest amount of code. Since they don't know you've done it in one line, they come back with three or four lines of code. You then get a chance to smugly show them your one-liner, thus proving beyond a doubt that you're the smartest kid on the block. The obvious beneficial effect of this activity is that you lose your friends very quickly, thus gaining valuable time needed to develop more one-liners.

Okay – what does all this have to do with using and abusing the DEF FN statement? The important thing to realize is that one- liner syndrome is a very dangerous programming practice. It's certainly fun, and sharpens your programming skills, but tends to produce code which is very hard to debug (even by you), and often runs slower than its multi-line equivalent (because of the contorted logic you need to use to accomplish the most in a single line). My point is that defined functions in interpretive BASIC are the single exception to this rule. This is because speed often isn't a major factor in business programs run under an interpreter (as discussed above), and as far as debugging goes, once the function is written, you use it the same way you use an assembler routine – just make sure it's adequately documented, and you can transfer it from program to program with great savings in development time.

Microsoft BASIC provides powerful tools for this type of function definition. You can include as many arguments as you like in a function, and most importantly, the arguments (and the function itself) may be character strings. You can also refer to other functions and "global' variables (that is, variables other than the function's arguments) within the function. This doesn't quite give you the power of multi-line functions, but you'd be surprised at what a single DEF FN can give you if you exercise a little creativity.

Without further ado, here are a few functions which you might find useful.

```
DEF FNROUND(X) = INT( X + 0.5 )
```

This function rounds its argument off to the nearest integer. Microsoft BASIC version 5 will do this for you whenever you assign a floating point value to an integer variable.

```
DEF FNROUND2(X,N) = INT( X / 10↑N + 0.5 ) * 10↑N
```

This will round X to the N'th decimal place. So FNROUND2(X,0) is the same as FNROUND(X), FNROUND2(X,2) rounds to the nearest hundred, and FNROUND2(X,-1) rounds to the nearest tenth. When using this and other numeric functions, bear in mind that they don't check for BASIC's precision limitations.

```
DEF FNROUND$(X) = STR$( INT( X + 0.5# ) )
```

Here's our first string function. It does more or less the same thing as FNROUND(X), except the result is returned in a character string. Also, "0.5#' is expressed as a double-precision constant to give us added accuracy of result. This, of course, is at the expense of execution speed, since extra conversions are taking place.

```
DEF FNROUND2$(X,N) = STR$( FNROUND2(X) )
```

Assuming we've defined FNROUND2 as above (make a similar assumption in all examples below), this function returns the same result in a string. It's an example of building functions using stuff we've already defined.

(...keep reading, we'll get to the ZX-80...)

```
DEF FNV(R$,S,L) = VAL( MID$( R$, S, L ) )
```

(continued next page)

This function is intended as a documentation aid in business programs. It is intended for use when we're processing data file records, and must extract numerical values from arbitrary points therein. It's not always effective to use the FIELD statement for this purpose, especially when we have many complex record types intermixed in the same file. The function extracts a string from R$, starting at S for length L, and returns its numerical value. When used in a program, its use might look like this:

```
ON FNV( EMPLOYEE$, STATUS, 2 ) GOTO 100, 200, ...
```

Note that the VAL function will not perform any error checking, so be VERY sure that the data that FNV processes is numeric.

```
DEF FNJUST$(A$,L) = RIGHT$( SPACE$(L) A$, L )
```

This function will right-justify a character string in a field of length L. It is most useful in combination with other functions, and produces much more compact source code than can be obtained with the RSET statement.

```
DEF FNV$(X,L) = FNJUST$( FNROUND$(X), L )
```

Here we have the complementary function to FNV. This function takes a value and a field length, and returns a right-justified, rounded value, ready to be placed into a data record which doesn't use FIELD statements.

```
DEF FNS2$(X) = MID$( FNROUND$(X), 2 )
```

This is a utility function for FNS (defined below). It returns the rounded value of X as a character string, leaving out the leading space or minus sign (Remember that STR$(57) = "b57", where "b' represents a single space).

```
DEF FNS$(X,L) = FNJUST$( MID$( "<b", 2 + ( X < 0 ), 1 )
                        + FNS2$( X )
                        + MID$( ">b", 2 + ( X < 0 ), 1 )
                        , L )
```

Whew! And you thought they were all gonna be easy! This function was developed for an application where negative numbers had to be surrounded with angle brackets, and all numbers had to line up in columns whose lengths were to be specified at execution time. Clearly, "PRINT USING" would be cumbersome here, and anyway, the columns had to be stored in text form on disk. Here's how the function works:

First of all, look at the FNS2$(X). This gets us the absolute value of X (rounded to the nearest integer) in a character string with no surrounding spaces. The MID$ functions supply angle brackets or spaces (read "b" as blank in this function) like this: if X is negative, then Microsoft BASIC will return −1 for the expression "X < 0." Thus, "2+ (X < 0)" will equal 1, so the first MID$ will return "<" and the second will give ">". If X is positive or zero, "X < 0" will be zero, so "2 + (X < 0)" will equal 1, and both MID$ functions will yield a space. We surround FNS2$(X) with both MID$'s, and then send the whole mess through FNJUST$ with field length of L to get a nice, neat, columnar result. Pretty impressive for BASIC, no?

```
DEF FND(D$) = ( FNV(D$,7,2) - BASE ) * 24
            + ( FNV(D$,1,2) - 1 ) * 2
            - ( FNV(D$,4,2) >= 15 )
```

This function was developed for a Credit Line Maintenance system which handled payables in semimonthly periods. The first period for each month was from the 1st to the 14th, and anything thereafter fell into the second period. A base year was defined as a two-digit number (i.e., "80") before FND was used in the program. The purpose of this function is to convert a date in "mm/dd/yy" format to a number which indicates the number of periods that have elapsed since January 1st of the base year. This makes complex date calculations (which were performed all over the program, in order to handle multiple credit lines with differing characteristics) quite simple.

The function first gets the year's value (FNV(D$,7,2)), subtracts the base year to determine the number of years elapsed, and multiplies by 24 to get the number of elapsed periods until January 1st of the indicated year. Then the month (FNV(D$,1,2)) is adjusted down (to get 0-11) and doubled, because there are two periods in every month. We add this value in to get the number of elapsed periods until the first of the indicated month. The last component is the most interesting: We get the day ("FND(D$,4,2)") and compare it to 15. If the day was less than 15, "(FNV(D$,4,2) > = 15)" will equal zero, so our partial result will stand. Otherwise, "(FNV(D$,4,2) < = 15)" will equal −1, so subtracting this from our partial result will cause the result to be incremented, to indicate a period in the second half of the month.

(...we're almost there...)

```
DEF FNDM$(D) = RIGHT$( STR$( 101 + (D MOD 24)/ 2 ), 2 )
```

We're now developing the complementary function to FND, in stages. FNDM$ takes a period number (computed by FND), and returns a 2-character string consisting of the month represented by the indicated period. "(D MOD 24)" gets rid of the year information in D. We then integer-divide by 2 to get a month number from 0-11. Adding 101 accomplishes two things: The 1 is added to adjust the month back to 1-12, and we add 100 to make sure that we get a zero in the the next-to-last digit for months 1-9. We then take the last two characters of the string value to get our final result.

```
DEF FNDD$(D) = RIGHT$( STR$( 101 + 14 * (D MOD 2) ), 2 )
```

This function returns a 2-character string containing the first day of period D. "(D MOD 2)" returns 0 for the first period in each month, and 1 for the second period. We multiply by 14 to get 0 for the first period and 14 for the second. We then proceed as in FNDM$ to get either "01" or "15."

```
DEF FNDY$(D) = RIGHT$( STR$( 100 + BASE + D \ 24 ), 2 )
```

Here's the year function. We integer-divide the period by 24 to get the number of years elapsed. This is added to our base value (described above). 100 is added to take care of leading zeros, and we then extract the trailing two characters of the result.

```
DEF FND$(D) = FNDM$(D) + "/" + FNDD$(D) + "/" + FNDY$(D)
```

This function puts everything together by constructing an 8-character string in mm/dd/yy format. It doesn't look that bad in this form, but consider the full definition without the use of auxiliary functions:

```
     DEF FND$(D) = RIGHT$(STR$(101+(D MOD 24)\2),2)+"/"
                   +RIGHT$(STR$(101+14*(D MOD 2)),2)+"/"
                   +RIGHT$(STR$(100+BASE+D\24),2)
```

This is perfectly legit... It's just a lot harder to read!

```
     DEF FNZ(P) = P * -( P >= 0 )
```

This is an example of doing simple conditionals through functions, rather than IF statements. It returns zero if P is negative, and P if P is positive. For example, consider the following segment of code which produces a simple graph:

```
     100  FOR J=1 TO N
     110  PRINT A(J);
     120  K = 20
     130  IF A(J) >= 0 THEN K = K + A(J)
     140  PRINT TAB(K); "*"
     150  NEXT J
```

Given the above definition for FNZ, we could rewrite this code as follows:

```
     100  FOR J=1 TO N
     110  PRINT A(J); TAB( 20 + FNZ(A(J)) ); "*"
     120  NEXT J
```

This approach, while succinct in terms of source code, will almost always cause your program to execute more slowly. In other words, use the technique sparingly.

Well, I hope I've given you some ideas about using BASIC functions in new ways. If you come up with any really neat functions that you'd like to share with other *Lifelines* readers, send them to me care of this magazine. I'll put together a follow-up article, thus obtaining fame, glory, and riches beyond imagination (well, two out of three ain't bad) for those who submit the best functions.

(...AND HERE WE ARE!)

Still got a terminal case of one-liner syndrome? Defining neat functions doesn't quite do it for you? Okay, bunky, have I got a contest for you! For want of a better name, we'll call it the
"OKAY, ALREADY, I'LL GET SOME
OF THE JUNK OUT OF THE HOUSE" contest.

Here's how it works:

Rule 1: Stay up all night and develop the best Microsoft BASIC (Version 5) one-liner you can. Any night will do.

Rule 2: Send your entry, along with a description of what the one-liner does, to me care of this magazine.

All entries must be independent of the terminal that is used. They must be postmarked no later than three days and four hours before the final judging date, which will be whenever I feel like it (let's say about 2 months). Entries will be judged on the basis of three factors (yet to be determined) by a completely subjective judging organization consisting of yours truly.

The prize: A SINCLAIR ZX-80 KIT!!!

That's right, folks, we're talking about the original ZX-80 kit, not this new-fangled ZX-81 baloney. The kit is in its original box (if I can find it), and I'm pretty sure it's complete (all I did was look at it - honest!)

The catch: I get to publish any and all entries in a future article or articles. All authors will be credited in the article(s), but only one ZX-80 will be awarded. Happy one-lining!

Change of Address

Please notify us immediately if you move. Use the form below. In the section marked "Old Address", affix your *Lifelines* mailing label — or write out your old address exactly as it appears on the label. This will help the Lifelines Circulation Department to expedite your request.

New Address:

NAME

COMPANY

STREET ADDRESS

CITY STATE

ZIP CODE

Old Address:

NAME

COMPANY

STREET ADDRESS

CITY STATE

ZIP CODE

An Overview of CB80

Bill Burton

About CB80

CB80 was first released last September by Compiler Systems, Inc. as an upwardly compatible adjunct to [their] CBASIC. CBASIC is a reliable and widely accepted pseudo-compiled language which has been used successfully in a variety of commercial applications. Nonetheless, some users have complained about CBASIC's slow execution. By contrast, CB80 is a true compiler which produces much faster code than CBASIC while offering several useful extensions. Note: Compiler Systems has since been acquired by Digital Research, Inc. Presently, CBASIC and CB80 are Digital Research products.

Last July, prior to CB80's release, I attended a seminar in Pasadena, hosted by Compiler Systems. One of the major purposes of this seminar was to introduce CB80 to authors and vendors of CBASIC application software. Preliminary specifics of CB80's design and performance were discussed at length by Gordon Eubanks, the author of CBASIC and originator of CB80, and by Paul Lancaster, whose expertise is evident in many of CB80's optimizations.

As described during the seminar, the design goal of CB80 was a true compiler, largely source compatible with CBASIC, which would typically require less [compile and linkage] time to produce smaller and faster programs than functional equivalents created by either Digital Research's PL/I-80 or Microsoft's BASIC-80 compiler. Obviously, this would translate to huge improvements over CBASIC's performance. At the conclusion of this review, I shall comment on how well CB80 has fulfilled the ambitions of its designers.

The CB80 Programs

CB80 consists of the following programs:

CB80.COM	(6K)	- Main Compiler Module
CB80.OV1	(13K)	- Compiler Overlay # 1
CB80.OV2	(13K)	- Compiler Overlay # 2
CB80.OV3	(16K)	- Compiler Overlay # 3
CB80.IRL	(20K)	- CB80 Support Library
LK80.COM	(6K)	- CB80 Linking Loader
CBCK.COM	(10K)	- Check/Patch Utility

The first four of these modules function as a reasonably efficient three pass compiler. Error detection and reporting is impressive. It is quite easy to isolate source code errors from the compiler output. The linking loader is remarkably fast.

It is important to note that the CB80 library has been implemented as an indexed relocatable [IRL] file. This arrangement, similar to that of PL/I-80, allows selected routines to be linked without requiring the entire library to be searched sequentially. Linking compiled code to standard relocatable libraries such as BASLIB.REL and FORLIB.REL (of Microsoft BASIC and FORTRAN) is generally a slower process involving considerably more disk activity.

CBCK (called CK80 in some releases) is described in the licensing guide rather than the manual and its purpose is somewhat unclear at this point. CBCK checks the integrity of the files on a diskette and is also intended to simplify making patches to CB80 programs. Should this be required, further information will be provided.

Conspicuously absent from the CB80 disk is XREF.COM, the cross reference utility, included with CBASIC. CB80 users who have access to copies of CBASIC may use XREF but this may produce confusing results unless their CB80 programs are written entirely in a CBASIC compatible command subset. Those converting from CBASIC to CB80 should be able to use XREF without problems. Still, XREF should be be rewritten for use with CB80 and added to the package in the near future.

RMAC and M80 are two of the most popular and capable macro assemblers currently available. These are included as part of PL/I-80 and the BASIC-80 compiler packages respectively. CB80 does not yet include a companion assembler nor does it include a library manager. I feel that a high level development tool such as CB80 should include both.

Using CB80

The CB80 compiler produces relocatable modules from ASCII source programs. One or more relocatable modules are linked with the CB80 library to form an executable program or optional overlays.

The example below illustrates the simplest case of using CB80 to produce executable code from an ASCII source program, 'test.BAS'.

```
Input file    : test.BAS
User enters   : CB80 test
Producing     : test.REL
User enters   : LK80 test
Output file   : test.COM
```

Overlays are required if a program will CHAIN to other programs. The procedure requires compiling a 'root' program with one or more overlay programs. Root programs are typically menu programs or programs which initialize COMMON storage and then chain to a menu. Let us consider the more complex case of compiling and linking 'test.BAS' and two overlay files, 't1.BAS' and 't2'.BAS.

```
Input files   : test.BAS, t1.BAS, t2.BAS
User enters   : CB80 test, CB80 t1, CB80 t2
Producing     : test.REL, t1.REL, t2.REL
```

```
User enters    : LK80 test (t1) (t2)
Output files   : test.COM, t1.OVL, t2.OVL
```

Up to forty overlays may be linked to a root module by a single command line.

Converting CBASIC Programs

Addenda to the CB80 manual document the few known areas of incompatibility. Included in the addenda pages is a prominent warning concerning use of the 'POKE' command, especially POKEs to location 110H, (which adjusts CBASIC's console width).

CB80 assumes infinite console width when executing PRINT statements terminated by a comma or colon. Carriage return linefeed sequences are not generated automatically when assumed console width has been exceeded. This change has been implemented to facilitate direct cursor addressing. Wherever this change may cause problems, the POS statement can be used. For example:

```
    CBASIC
    FOR I% = 1 TO 1000
    PRINT I%,
    NEXT I%

    CB80
    FOR I% = 1 TO 1000
    PRINT I%,
    IF POS > 64 THEN PRINT
    NEXT I%
```

Owing to the increased speed with which CB80 performs integer operations, integer timing loops written for CBASIC will not produce adequate delay. For example, a CBASIC program might display a message on the screen for a few seconds like this:

```
PRINT "THE MESSAGE"
FOR I%=1 TO 1000
NEXT I%
... clear the screen - print new message
```

CB80 would also print the message but to the eye it would appear only as a glitch on the screen. CB80 code to replace this CBASIC timing loop might appear as follows:

```
PRINT "THE MESSAGE"
FOR J%=1 TO 50     \ Real values can be used
FOR I%=1 TO 1000 \ instead of nested loops
NEXT I%
NEXT J%
... clear the screen - print new message
```

Note that the two NEXT statements in the example above must appear on separate lines (this was not a requirement of CBASIC). This restriction encourages the programmer to indent nested loops for clarity, and as such should be considered a design improvement.

The CHAIN statement introduces a minor incompatibility. All CBASIC programs have the form, 'filename.INT' and all CHAINS in CBASIC are to INT type files. CB80 requires a 'root' program, 'filename.COM' and, optionally, one or more overlay programs, 'fname1.OVL', 'fname2.OVL' etc. When converting CBASIC programs to CB80, INT filename extents will have to be changed to either COM or OVL.

In rare cases, FOR-NEXT loops may cause conversion problems. Most BASICs, including CBASIC, do not evaluate FOR-NEXT until the loop has executed at least once.

```
    FOR I% = 1 TO -1
    PRINT I%
    NEXT I%
```

Most BASICs will print a '1' on the screen when executing these lines. CB80 will not display anything. Although this example might seem to point out possible compatibility problems, CB80 should not be faulted. The logic of FOR-NEXT loops should be evaluated before they are entered.

CB80 implements all of CBASIC's commands except for FILE and SAVEMEM. The function of the FILE command may be synthesized as follows:

```
    IF SIZE (FILENAME$) <> 0 \
    THEN OPEN FILENAME$ AS FILE.NO% \
    ELSE CREATE FILENAME$ AS FILE.NO%
```

SAVEMEM is used in CBASIC to reserve space in memory for assembly language routines. This command has no meaning in CB80 because the location of assembly language routines is assigned by the linking loader.

CBASIC allows a single identifier to be used as both a simple and subscripted variable name in the same program.

```
    DIM (A%) 10
    A% (1) = 10
    A% = 100
```

This is valid in CBASIC but it will produce a fatal CB80 compiler error. Again, this seeming restriction obliges programmers to write more logical and readable code and should therefore be considered a design improvement.

Strings in CBASIC are restricted to 255 characters. CB80 allows strings up to 32K. For this reason, CB80 requires an extra byte for internal representation of string lengths. To avoid logical errors, CBASIC code in which the SADD function is used with PEEK and POKE to pass strings to assembly language routines will have to be rewritten.

```
    CBASIC
    LEN% = PEEK (SADD(STRING$))

    CB80
    LEN% = (PEEK (SADD(STRING$)) AND 07FH \
         + PEEK (SADD(STRING$) + 1)) * 256
```

CB80's READ and INPUT statements handle integers differently. In CBASIC, all numeric values are accepted as real quantities and converted to integers when required. CB80 accepts integers directly but stops conversion at the first non-

(continued next page)

integral character.

```
DATA 10.7, 1E2
READ A%, B%
```

After this READ statement had executed, the values of A% and B% would be:

```
CBASIC          CB80

A% = 11         A% = 10
B% = 100        B% = 1
```

CB80 uses a signed binary value to represent the amount of free memory returned by the FRE statement. This will result in a negative value if there are more than 32767 free bytes.

CB80, The New Features

CB80 supports alphanumeric labels which may be used instead of line numbers to increase program readability. For example:

```
CBASIC
GOSUB 19003

CB80
GOSUB UPDATE.CLIENT.HISTORY.FILE:
```

To some degree, use of alphanumeric labels can make a program self-documenting, thereby reducing the number of remarks needed to clarify program logic. Note that CB80 requires a colon as the last character of an alphanumeric label.

Variable types may be declared at the beginning of a program. In some cases this allows a functionally identical program to be entered with many less keystrokes because, once declared, integer and string variable references need not be followed by a type identifier, (string$ or integer%).

CB80 programs written for MP/M will likely require use of CB80's LOCK and UNLOCK commands to supervise file access. LOCK and UNLOCK dictate whether multiple users can share the same file and also prevent multiple users from attempting to update the same record simultaneously. In CB80, files are assumed locked if opened with the CREATE statement and unlocked if opened with the OPEN statement.

The ATTACH statement returns -1 (logical True) if a printer is attached.

```
TRUE% = -1 : FALSE% = NOT TRUE%
ATTACH (logical.print.device%)
IF ATTACH THEN GOSUB HARDCOPY.ROUTINE
```

CB80 allows IF statements to be nested.

```
IF I < J THEN    \
  IF K > L THEN \
    X = 3        \
  ELSE \ this ELSE matches second IF
    Y = 2
```

Some CBASIC users will probably be delighted that this feature is finally available. Personally, I feel that using nested IF statements makes code less readable.

Almost anyone who has used the newer Microsoft BASICs will be pleased to know that MOD, INKEY and ON ERROR (with ERR and ERRL) have been implemented in CB80. Also, PUT and GET allow binary data to be written to or read from files, a single byte at a time.

PUBLIC and EXTERNAL are used to create user defined functions which, once compiled, may be linked with other modules in the manner of intrinsic library functions. Proper use of PUBLIC and EXTERNAL references is somewhat complex and explanation is best deferred to the CB80 manual.

Disadvantages Of CB80

It appears that CB80 uses some extended functions of CP/M 2.x and will not run under CP/M 1.x or lookalikes. This may be of concern to software vendors whose established market includes a fair number of users with unsupported hardware.

Debugging usually accounts for much of the time spent in program development. In this regard, CB80 appears somewhat deficient. CBASIC's TRACE utility has been dropped. I suspect that many CB80 users will find this a significant shortcoming.

Many identical library routines must be linked redundantly into every overlay. Under some circumstances this can waste lots of disk space. By contrast, both CBASIC and the BASIC-80 compiler produce smaller modules which require roughly 16K of shared overhead (CRUN2 and BRUN, respectively). However, because CB80 does not use shared run time support, individual programs or overlays will run in less memory, (although STAT will report that they are larger).

To sell or distribute CB80 composite programs, (those programs incorporating library routines), one must first acquire a license from Digital Research which costs $2000 annually with the possibility of a 20% escalation for each of the first four renewals. The licensing fee applies regardless of [sales] volume or the number of discrete CB80 products involved. This policy may prove a major deterrent to new software firms as well as smaller firms and those developing products for specialized use.

CB80 Performance And Reliability

I have run several benchmark tests to compare CB80 with CBASIC and the BASIC-80 compiler. As expected, CBASIC is no speed contender. Floating point operations are considerably faster with the BASIC-80 compiler than CB80. This is largely a meaningless comparison because floating point arithmetic of the BASIC-80 compiler is performed as single precision binary as opposed to the fourteen digit BCD representation used by CB80. (BCD is an acronym for Binary Coded Decimal, which is an inherently more accurate but slower system). CB80 enjoys a substantial edge in speed of integer operations and a slight speed edge in performing string manipulations. Benchmarks of larger programs suggests that, overall, CB80 will provide slightly faster execution than the BASIC-80 compiler and much faster execution than CBASIC. Note: Although I did not include PL/I-80 in these particular benchmarks, earlier testing indicates that its performance is roughly comparable to the BASIC-80 compiler.

Two known bugs are acknowledged in the addenda. INKEY will not work with CONCHAR% due to a bug in CP/M 2.2 and multiple line defined functions which return strings may not be referenced twice within the same expression.

```
    DEF FNA$
    FNA$ = A$
    IF LEN (B$) > 10 THEN RETURN
    FNA$ = A$ + B$
    RETURN
    FEND

    PRINT FNA$ + FNA$   <--- ERROR!
```

Otherwise, version 1.2 appears to work correctly. However, CB80 is still a relatively new product and it is quite possible that minor bugs persist. At this point, CB80 has been user tested in enough different applications so that bugs which have not yet been found are apt to be trivial.

Digital Research has an enviable record of providing committed product support on a timely basis. This suggests that any necessary fixes, refinements or enhancements to CB80 will be completed quickly and professionally. Despite the fact that I first acquired CB80 version 1.0 from Compiler Systems, Digital Research has sent me updated versions without charge (and before I had contemplated preparing this review).

CB80 is a useful and well conceived product which has already gained acceptance amongst many demanding users for development of complex applications. Most CBASIC programs can be transported easily, and CB80 is fast! My principal reservations about [version 1.2 of] CB80 center on the lack of a companion macro assembler, library manager and cross reference utility. Any or all of these will be welcome additions to future releases.

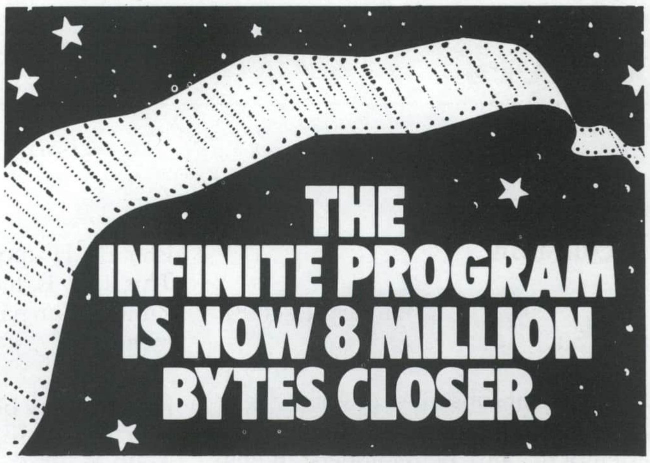
In conclusion, despite the few flaws and omissions (which are typical of reasonably new products), I highly recommend CB80 to any prospective user who doesn't find the licensing terms too restrictive.

T/MAKER Users Group

The T/MAKER Users Group will provide members with a bimonthly newsletter, where they can share experiences in the areas of text editing, financial modeling, personal and business accounting, mathematical and statistical applications, and other fields of interest. Users' technical and non-technical questions and problems will be answered.

The fee for a one-year membership is $9; those considering membership may request a free sample copy of the newsletter. The following information, along with your name and address, should be included with membership checks: computer name, model and memory; terminal manufacturer and model; disk format; operating system name and version; T/MAKER version; areas of interest

Send your requests to T/MAKER Users Group, 2801 Flagmaker Dr., Falls Church, VA 22042.



INTRODUCING PLINK II,™ a giant step towards microcomputing perfection: the eight million byte program.

Lifeboat's fast, versatile PLINK II two-pass linkage editor helps you create large, sophisticated programs without any programming changes.

PLINK II is unique. While other linkage editors require you to construct programs in memory, PLINK II constructs your program on disk. You gain complete use of your computer's memory, using the full address space, up to 64K resident segments even on smaller machines.

PLINK II also constructs applications larger than your computer's address space. By creating arbitrarily complex overlay structures, you can flexibly bring in chunks of code from disk during execution.

Ideal for use with all popular high level languages, PLINK II even simplifies construction of bootstrap routines, allows menu-driven links, and more.

Reach out towards the infinite through PLINK II. . . and see how far your programs can take you.

Lifeboat Worldwide offers you the world's largest library of software from its offices in the U.S.A., Japan, U.K., Switzerland, W. Germany, and France.

Mail coupon to: Lifeboat Associates, 1651 Third Avenue, New York, New York 10028 or call (212) 860-0300. TWX 710-581-2524 (LBSOFT NYK)

☐ Please send me more information on PLINK II.
☐ Please send me a free Lifeboat Catalog.

Name ______________ Title ______________

Company ______________

Street ______________

City ______________

State ______________ Zip ______________

#1 SOFTWARE WITH FULL SUPPORT

PLINK II is a trademark of Phoenix Software Associates, Ltd. Copyright ©1981, by Lifeboat Associates.

Lifeboat Associates
World's foremost software source

Letters

A Question

January 11, 1982

Dear Sir:

I am a recent subscriber to your magazine and find it invaluable toward my attempt to gain a better understanding of the CP/M operating system. I need to know more about its inner secrets and I think I have found a good source in your magazine. I am currently using CDOS and am running into trouble because it does not properly support modern CP/M software on the market.

I have been attempting to locate a version of CP/M which I can run with my CROMEMCO 16FDC floppy disc controller board. I would also like to expand it to work with one of the hard disc systems on the market but need to retain the 16FDC and my 8 inch disks for backup. Can you steer me toward a supplier who might already have a version of CP/M that might fill my needs? I have written to Digital Research but they sent me a list of suppliers but so far I have not been successful in getting responses to my letters when I write to those suppliers.

Sincerely,
Howard O. Ehlers

Can anyone suggest a solution to Mr. Ehlers' problem? – *Editor*

On BASCOM and CDOS

January 12, 1982

Dear Sirs:

In "Tips & Techniques" (January 1982) Roberto Denis wrote concerning the BASCOM problem with CDOS. He states that "an *undocumented* feature of CP/M is that it also returns in the A register whatever is returned in the L register ...".

This is *not* undocumented. On page 3 of the "CP/M 2.0 Interface Manual", Digital Research states "for reasons of compatibility, register A=L and register B=H upon return in all cases". This fact is mentioned again in the summary on page 46 of the same document.

Roberto may be correct in his conclusion that Microsoft could easily fix the problem. But so could CDOS [sic]if they wanted to extend their compatibility with CP/M. In any event, Microsoft's use of the A register is not outside the documented features of CP/M.

Sincerely,

William R. Brandoni
Willoughby, Ohio

Praise for Christensen

January 18, 1982

Dear Ward,
I have been thinking of writing you for some years now, but just haven't taken the occasion to get "pen in hand" until now. I've tried to reach you several times while I have been in Chicago on business trips but each attempt was singularly unsuccessful. I have been using some of your software quite frequently over the last few weeks and the thought keeps coming back.

I just had to write to let you know just how much I appreciate the contributions you have made to the micro field and to me. I don't know just how often you get letters of thanks but it should happen quite frequently. All of the S-100 system users in the Toronto area here that I know use many of your packages in the daily pursuit of their hobby.

The first time that I came across your work, you had written an article for Dr. Dobbs. It described some of your efforts in getting a new disk system going, I think it was with Micropolis drives, and a Disassembler. That disassembler is still being used by some people here as are a multitude of your other 'works'.

Those which I appreciate most are the MODEM, XMODEM and DU series of applications. I don't know what I'd do without them. I don't have the necessary expertise to sit down to write them yet but am able to achieve my immediate goals through their use. I intend, one of these days to put another of your products to work, specifically, your assembler-written CBBS package.

That's all.... Just: - Thanks a bunch! If you haven't received any other similar letters from the guys here in Toronto, thanks a bunch more for them, too.

Sincerely,
Bill Harnell
Scarborough, Ontario
Canada

International Standards

15/1-82

Some suggestions for the Editor:

CONTROL CODES FOR SCREEN HANDLING

I am sure that many newcomers to application packages become confused when they try to run an install program on a screen type not referenced in the install package. Some install programs are more helpful than others with either hard-copy documentation or screen prompting – cf. T/MAKER II.

Could *Lifelines* try to gather information about control codes used on the various screen types and publish this in tables? This would then allow a person unable to install a program to compare the screen he/she is using with others and then perhaps find one that works in a similar manner.

ASCII - DATES - DECIMAL POINT

ASCII does not take account of characters found in other world languages. This is a particular problem in Europe and what is needed here is a EUROPEAN STANDARD CODE FOR IN

FORMATION INTERCHANGE (ESCII) - allowing for a character set including system control codes of up to 256. Take for instance the German letters: ä, ü and the double ß - or the Norwegian ø, æ, or å.

The problem is not only one of graphic representation, but also their hierarchy in for instance a sort program.

Another problem is that most application packages operate with an American method of dating - 1/15/82 for January 15th, 1982. But even in Scandinavia there are differences between Norway and Sweden in the presentation of the date. In Norway we would expect 15/1-82 whereas in Sweden they would write 82/1/15 - i.e., the year first.

The decimal point also presents problems. Where in the US and UK one uses a period (.) - in many other countries standard practice is for the comma (,) to represent the decimal divider and the period (.) the thousands separator.

As many of the world's software houses are located in the USA and since the major operating systems also originate there, little account seems to be taken of requirements elsewhere. I would appreciate the Editor bringing these matters to the attention of the vendors via the columns of your esteemed publication.

Yours sincerely,
Brian J. Brown, System Applications Mgr.
Scanword, AS
Oslo, Norway

Notice

The February issue was placed into the mail on January 27th. 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 March 1982, into the mail around February 25th. We will print each month the date of the previous issue's mailing and would appreciate your help in tracking the deliveries.

Bugs and Patches FABS, FABS II, and BASIC-80

Bill Norris

FABS-II does not work with Version 5.21 of the BASIC-80 Interpreter. The two do not link because of a change in the new MBASIC. One way to solve this problem is to preserve your CPU's registers while MBASIC is executing. This way you can go back to the system, SAVE MBASIC with FABS and have the SAVE program restore the registers. Use this until a new version of FABS solves the problem. This patch should also be useful for ULTRASORT.

```
;*****************************************************************
;**                                                             **
;**     Program name is FIXBAS.MAC                              **
;**     Program to permit reentry to MBASIC 5.21                **
;**     This is not needed with previous versions of            **
;**     MBASIC.  (It may be used nevertheless).                 **
;**     Originally written to permit saving a memory            **
;**     image of MBASIC/FABS.  Useful for ULTRASORT too.        **
;**                                                             **
;*****************************************************************
;**                                                             **
;**     To use with MBASIC 5.21 and FABS/FABS-II:               **
;**     1>  Assemble                                            **
;**             With M80, the command line could be:            **
;**                     A>M80 8000,=8000                        **
;**     2>  Produce a HEX file                                  **
;**             With L80, type the following:                   **
;**                     */p:8000,8000,8000/n/x/e                **
;**     3>  Load the code with DDT                              **
;**                     A>DDT 8000.HEX                          **
;**                     -G0                                     **
;**     4>  Run RELFABS/RELFABS2                                **
;**                                                             **
;**     5>  Load MBASIC                                         **
;**                     MBASIC /M:xxxx (see note 1)             **
;**     6>  Execute 8000 patch                                  **
;**                     A=&H8000 (see note 2)                   **
;**                     CALL A                                  **
;**                     ( you will now be back in CP/M          **
;**     7>  Save composite program.                             **
;**                     A>SAVE xxxx PROGRAM.COM (see note 1)    **
;**                                                             **
;*****************************************************************
;**                                                             **
;**     Date: Feb. 4, 1982.    Written by: Bill Norris.         **
;**                                                             **
;**     NOTE 1:  the values xxxx are specified by RELFABS       **
;**                                                             **
;**     NOTE 2:  if your computer has very little memory,       **
;**              this code will probably be overlaid by         **
;**              FABS.  You may try locating it lower (the      **
;**              current MBASIC ends at 6000 hex) or higher.    **
;**              If you choose to set the origin higher, make   **
;**              sure that it lies completely above the BIOS.   **
;**                                                             **
;*****************************************************************

        aseg
start   equ     8000h   ; This value is not critical.  It
                        ;   must, however lie somewhere
                        ;   above MBASIC and below FABS,
                        ;   or completely above the BIOS.
```

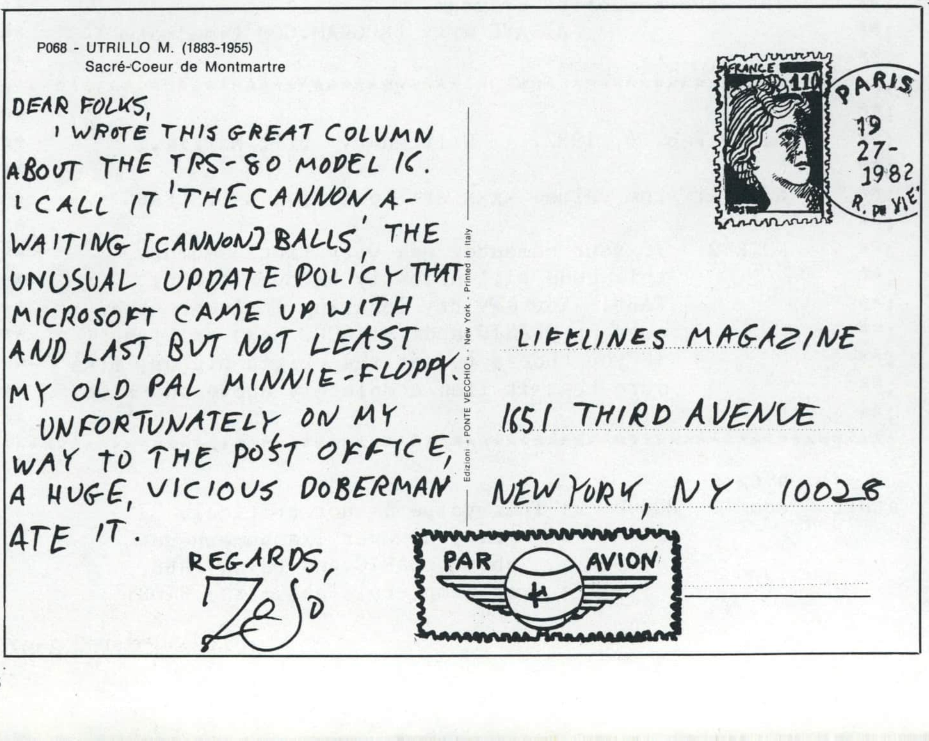
(continued next page)

```
        org     start
        shld    hiho            ; Save HL.
        lxi     h,0             ;
        dad     sp              ; Current stack.
        shld    olstak+1        ; Save it.
        lhld    hiho            ; Get back HL.
        lxi     sp,nustak       ; Push registers here.
        push    h               ; Save HL.
        push    d               ; Save DE.
        push    b               ; Save BC.
        push    psw             ; Save AF.
        lxi     h,0             ;
        dad     sp              ;
        shld    restak+1        ;
        lxi     h,restak        ; Force jump to code which
                                ;   restores the machine state.
        shld    101h            ; Patched.

        jmp     0               ; Go to CP/M and issue the save
                                ;   command specified by RELFABS.


restak: lxi     sp,0
        pop     psw
        pop     b
        pop     d
        pop     h
olstak: lxi     sp,0
        ret
hiho:   db      'This space is for the stack!'
nustak: nop

        end
```


P068 - UTRILLO M. (1883-1955)
Sacré-Coeur de Montmartre

DEAR FOLKS,
I WROTE THIS GREAT COLUMN ABOUT THE TRS-80 MODEL 16. I CALL IT 'THE CANNON AWAITING [CANNON] BALLS', THE UNUSUAL UPDATE POLICY THAT MICROSOFT CAME UP WITH AND LAST BUT NOT LEAST MY OLD PAL MINNIE-FLOPPY. UNFORTUNATELY ON MY WAY TO THE POST OFFICE, A HUGE, VICIOUS DOBERMAN ATE IT.

REGARDS,
[illegible]

Edizioni • PONTE VECCHIO • New York • Printed in Italy

FRANCE 110

PARIS 19 27- 1982 R. DU VIE

LIFELINES MAGAZINE
1651 THIRD AVENUE
NEW YORK, NY 10028

PAR AVION

The Pipeline (continued from page 4)
have been working on a project to put a 6809 microprocessor on the Heath ET-3400 trainer. Our side of the project was to get the software working, while Don Burtis of Burtronix developed the hardware interface.

To do our task we needed some tools to blow PROMs and hopefully give us a chance to simulate the software before doing so.

While searching around for the Ideal tools, we ran across Microsoft's Assembly Language Development System (ALDS) for the Apple, and Vista's PROM Simulator System also for the Apple.

The Vista board, priced at $495, fits in any of the available I/O slots on the computer's backplane, and simulates PROM from RAM allowing any new code to be tested within the system before a PROM is burned.

The PROM Development System features a menu-driven program that operates under Apple DOS and is able to simulate and program: 2708, 2716, 2532, 2732, and 48016 EEPROMs.

The disk-based, menu-driven program development monitor, permits the direct loading of code from an assembler to the onboard 4K RAM for simulation purposes. The simulator section is made up of the memory, the Apple bus interface, and the cable interface used to plug into the target machine. The RAM is two CMOS 2K x 8 Hitachi 6116 RAMs, and is mapped into a 2K address space.

The interconnecting cable that is designed to plug in a 24-pin dip socket series is terminated with 39 ohms to reduce overshoot and ringing. The board is programmed for the type of PROM to be simulated by using wire-programmable dip plugs.

The PROM Developmen. ystem isn't restricted to developing fi.mware for the 6502 uP, but can be used to simulate or burn PROMs for any processor. In our case we used XASM-09 to develop the code.

Although the software monitor works with Apple DOS, you can employ Microsoft's $125 Assembly Language Development System to create code for Z-80, 8080, 8088, and 8086 systems, and load the simulator's RAM with the output. For our project, we just happened to do most of our development work on the hard disk system, downloaded it to the Apple and used ALDS to convert it to Apple DOS. This is only just one of the features of the software, and it performs the job admirably.

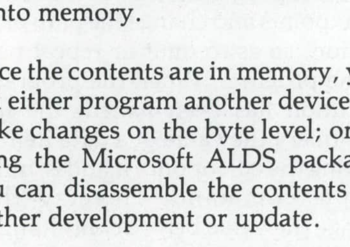
Once the code is developed and passes inspection, the PROM can be created by using the onboard Textool socket to plug in the PROM; then by selecting the Burn PROM routine from the menu, the programming can take place.

Besides serving as a simulator and PROM burning tool, the development system can be employed for diagnosing problems with existing PROMs, or for Copying PROMs. This function is a standard feature of the system monitor.

To copy a PROM all that is necessary is to plug the desired PROM to be read in the Textool socket and enter the read routine. This routine then reads the contents of the PROM and copies it into memory.

Once the contents are in memory, you can either program another device, or make changes on the byte level; or by using the Microsoft ALDS package, you can disassemble the contents for further development or update.

While working on this project, Pat and I came to several conclusions about the tools we used: one, the Vista board is worth the $495 just for the time it saves; XASM-09 is easy to use, handles the job quickly but unfortunately lacks macros; next, Microsoft's ALDS package belongs on every programmer's work bench, and like the simulator board saves a great deal of time and wasted effort. Should this sound like an endorsement of the above products – it is.



Macros of the Month

Edited by Michael Olfe

Here are some more contributions from Ward Christensen. We hope this will inspire some of you PMATE users out there.

— Mike Olfe

I like the "repeat" key in WordMaster, and missed it badly. So, I faked up a "repeat-4" key function to PMATE. I now can hit ↑W (the old WM repeat key), followed by any of several keys.

While this repeat-4 key "almost" made up for the lack of the repeat key, I still found there were times I used the repeat key in WM that it couldn't handle. For example, to put a line of '—' across to break things up, as I have done in this document. I selected a control-½ or control-], depending upon whether I am on the H-19, or on an old keyboard hooked to a VIO, as a "meta" (extended) control key. In "meta" mode, I have put in some extra functions

meta '-'	puts the 63 dashes on the screen
meta word right	deletes word right
meta word left	deletes word left
meta C/R	edits command line
meta scroll u.	goes to top of file
meta scroll d.	goes to bottom of file

Here are the user-instant commands I have added:

```
;
;192: Beginning/end of line:
UIC192  DB      '@X>0[0L][L-M@T=13[%]@T<224[M]]',0
;
;193: Erase entire line:
UIC193  DB      '0LK',0
;
;194: Erase to end of line
UIC194  DB      'K13I-M',0
;
;195: Open 4 lines
UIC195  DB      'I',13,13,13,13,esc,'-4M',0
;
;196: 4 scroll up
UIC196  DB      '-80L',0
;
;197: 4 scroll down
UIC197  DB      '80L',0
;
;198: 4 lines up
UIC198  DB      '@X,-4L@SQX',0
;
;199: 4 lines down
UIC199  DB      '@X,4L@SQX',0
;
;200: 4 right
UIC200  DB      '4M',0
;
;201: 4 left
UIC201  DB      '-4M',0
;
;202: 4 delete
UIC202  DB      '4D',0
;
```

(continued next page)

```
;203: 4 kill
UIC203  DB       'OL4K',0
;
;204: 4 word right
UIC204  DB       '4W',0
;
;205: Insert a line of dashes
UIC205  DB       '6[i----------',esc,']13i',0
;
;206: Display version number
UIC206  DB       'I11/30/81 18:50',esc,0
;
USRCOM                   ;INITIALLY EXECUTED USER COMMAND
        DB       0
,
```

Here are some more of my permanent macros:

DELETE TRAILING SPACES

I sometimes want to pick up a WordStar document, and convert it to PMATE. In formatted mode, WordStar puts spaces at the end of lines. PMATE doesn't. The following macro deletes trailing spaces:

```
^X 21-2m@t=32[d][1]
In detail:

^X                ;^X(space): space is the macro name
21                ;go down 2 lines
-2m               ;        then back over C/R, to
                  ;        possible space
@t=32             ;If the character is a space
        [d]       ;delete it
        [1]       ;otherwise move down a line
```

DELETE REDUNDANT SPACES

If I pick up program documentation, and find it to be justified, I sometimes want to delete the "noise" duplicate spaces justification inserted. This macro deletes all duplicate spaces, except those following a period.

```
^X.[s  $-3m@t=".{3m^}2mgOK?$@k=127[dqr]]
```

which means:

```
^X.               ;permanent macro named "."
[                 ;repeat
s  $              ;search for a space
-3m               ;back up 3 characters
@t=".             ;is the char a period?
        {3m^}     ;yes, move 3 and loop to "["
2m                ;otherwise move 2 spaces
gOK?$             ;print a prompt, wait for key
@k=127            ;if the key pressed is "DEL"
        [dqr]     ;then delete it and re-display
]                 ;loop
```

— Ward Christensen

New Products

The software described below is available from its authors, computer stores, software houses, distributors and publishers.

COBOL ANIMATOR
Micro Focus

This product is designed to display the source listing of a program on the screen, moving the cursor from statement to statement as execution proceeds. Programs can be run a statement at a time, or continuously so that the execution path can be observed. Speed can be adjusted under continuous animation. COBOL ANIMATOR can also be run normally; in this case only user displays are shown – it can be switched back into animation mode to examine, for instance, the program's operation with a certain data set.

The interactive debugging capabilities allow the programmer to set execution breakpoints and change the path of execution, so as to omit or repeat parts of the program. When the program's execution has been halted, the user may query the value of a data item by moving the cursor onto it and issuing a simple command. He may then change the value before continuing.

COBOL ANIMATOR may be used on any COBOL source program conforming to the ANSI 1974 standard, regardless of the original compiler used to write it.

COBOL SLIDESHOW
Micro Focus

This software tool is intended for the creation and maintenance of application packages. It allows the developer to present information in menus, with still or moving color graphics, or text displays. Linked with applications programs, it is designed to permit software which is more user-friendly. In addition, it can be manipulated without programming, thus separating the programming and application design functions in software development.

SLIDESHOW can also be used with already-existing software products. It treats an application as a number of discrete steps linked together by many alternative paths. The designer specifies a sequence in which application steps are performed by creating a control file, in which each record specifies a single step in the application. Any number of control paths can be built in to index and sequence system components.

SLIDESHOW is conceived as an operating environment which can enforce a logical pattern on the operation of applications packages when they are used by people having little background in computing.

Fin-Ratio
FPS Associates

This package is designed to aid Financial Ratio Analysis. Amounts from balance sheets and income statements are entered from the keyboard and Fin-Ratio generates eighteen financial ratios, forty two derived data and percentage relations. Sets of data from different balance sheets, income statements and sets of norms can be combined for financial analyses.

Twenty-nine business or company internal norms can be entered and printed out next to the financial data. A "z" ratio is generated, which is a composite indicator of propensity for financial failure. Amounts can be up to $99,999,999 thousand.

Fin-Ratio runs under CPM-80 and is available in Microsoft BASIC; it requires 48K RAM.

Nevada Edit
Ellis Computing

This character-oriented full screen video display text editor is designed specifically for computer program text preparation. It includes single key commands for cursor control, scrolling, block moves, search and replace, tab setting and multiple file insertions.

It requires CPM-80 and a minimum of 32K RAM.

Nevada Pilot
Ellis Computing

This version of PILOT (Programmed Inquiry Learning Or Teaching) is intended for both business and educational use. It is string-oriented, expressly for interactive applications like data entry, programmed instruction, and testing. PILOT allows the inexperienced user to quickly develop dialogue programs. PILOT can be used with BASIC, COBOL, and PASCAL to simplify training.

This package includes a full screen text editor, commands to drive optional equipment. It requires CPM-80, 32K RAM.

New Publications

CBASIC Users Guide
by Adam Osborne, Gordon Eubanks and Martin McNiff
This guide is an accompaniment to the CBASIC manual, providing more examples and tutorials. Chapters are devoted to input programming, output programming, file handling, and assembly language interface. The book should be most useful for those new to CBASIC who wish to run application software and write their own programs. Narrative descriptions of CBASIC commands, detailed explanations of error messages, and other features should help answer the questions of users.

New Versions

Nevada COBOL
Version 2.1

This update implements a large number of bug fixes, including the following:

1. Called programs with linkage and I/O address problems.
2. Two Random files in a program with different size keys.
3. Write in Random I/O mode to extend the file now works.
4. Called program not on the default drive now works.
5. IF A = B GO TO X ELSE ...(IF's with GO TO's now work.)
6. REDEFINES at 01 level in file section now are flagged as errors (ANSI '74).
7. IF A = B OR C = D GO TO X. (Compound conditionals with GO TO's now work.)
8. Records in working-storage over 4095 characters are now flagged as errors.
9. The printer must be opened as output only or an error is reported.
10. The compiler now accepts 4 or 6 (ANSI '74) position line numbers. It looks at the first line and determines which to use. The default is four positions.
11. Literals can now be followed by a comma. In some cases this was being incorrectly reported as an error.
12. The compiler expands all tabs every eighth position, i.e. 1, 8, 16, 24, 32.
13. The error report now lists user line numbers and internal line numbers; if an error occurs in a copy file it is flagged with a C.
14. The display series now shows each operand on the same line, per ANSI '74.
15. IF A (X1) IS NUMERIC and its subscript were incorrectly flagged as an error; this has been fixed.
16. The SELECT error which occurred when running under MP/M II has now been repaired.
17. The source file name is now output to the error report.
18. COMP and COMP-3 cannot be edited-receiving fields and are now correctly flagged as errors.
19. Compound Conditionals (IF A = B OR = C) will now handle the omission of the A-operand.
20. Addresses are now handled correctly when a CALLED program is CALLED by another CALLED program.
21. SYNC, BLANK WHEN ZERO, JUSTIFIED, USAGE can occur before PICTURE now.
22. -9(18) is now properly handled.
23. (Space . Space) ANSI '74 period can be preceded by a space now.
24. When certain combinations of edit symbols are used the compiler goes into a loop; this is now reported as an error.
25. Error report heading and other messages are now on file W5.CBL; error messages can now be on more than one line.
26. RENUMBER.CBL now flags short records and records with tab characters in the first four columns. It cannot expand tabs or number blank lines.
27. The rewrite of a variable file when the input was null set has been corrected.
28. Level numbers can now be 01

(continued next page)

through 49 or 77, one or two positions.
29. Random files up to 8 megabytes are allowed.

PAS3 Medical
Version 1.78
PAS3 Dental
Version 1.64

Two bugs have been corrected in these versions:

1. The aging report now prints the last person on the page or the top of the next page.
2. If the number of treatment details on regular billing and Insurance billing exactly matches the number of lines on the form the last treatment on the form does not print. The totals are correct.

PLAN80
Version 2.2

In version 2.1 there was a complete rewrite of the RULES processing logic; this resulted in some problems, which have now been fixed. A "?" may now be used for on-line input of values. The "↑" may be used for exponentiation. A Cell-to-cell assignment statement of the form

(ROWX,COLY) = @(ROWA,COLB)

now operates properly.

PLAN80 now runs on an Apple II with a standard 40-column monitor, in addition to Apples with 80-column boards or external terminals. However, 40-column Apples without a shift key modification are unable to generate lower case letters to control cursor movement in the DISPLAY mode. You may use the following alternative codes, which have been added:

Cursor Direction	Primary Key	Alternative Key
up	i	,
left	j	<
right	k	>
down	m	.

Both primary and alternate keys may be prefaced by a number to move the cursor by more than one row or column. Thus "12." would move the cursor down twelve rows.

Here are some other bug fixes:

1. The INSTALL program now interprets "+" as the start of a specification of null characters to be generated at the end of a screen control sequence.
2. Automatic page breaks for reports with a large number of columns now operate properly for the third and subsequent pages.
3. Blank lines generated by rows using the specification "S" now are counted for automatic page breaks.
4. The automatic form feed which formerly occurred at the beginning of a report printing now is executed at the end.

Series 9000 Medical and Dental Management System
Version 1.10

This version fixes the following two problems in Version 1.08:

1. When editing the patient master record from zero dependents to one or more, the system operates properly. If, however, the record which originally contained dependents was edited back to zero, the dependents were not deleted from the database, and some alphabetic reports would not operate properly, unless the Master record was deleted and re-entered once more, along with disk space recovery or end-of-month processing. This would not overwrite or disrupt any other records, but could be a possible problem until the dependents are deleted by the method mentioned above, or with a text editor.
2. ICDA Codes (Medical Systems Only) with modifiers could not be deleted once entered, only edited. Users of previous versions may safely use a text editor on MDICDA.DAT file and delete the desired code. This will not affect any other ICDA records or indexing.

In addition, a new American Dental Association form has been implemented.

Series 9000 Medical and Dental Management Systems
Version 2.0

This further update is scheduled for release on March 1st 1982. It comprises several enhancements to the software. A new programmer's section has been appended to the operator's manual, for data file layout and interfacing techniques. Sub-menus now have an "M" option for direct access to the main menu.

Inter-Office Administration features have been expanded to allow thirty characters in doctor and insurance address lines; this permits Series 9000 to accommodate the new zip codes. A second address line has also been implemented. The ICDA/CPT/ADA code description lines can also be thirty characters, and sixty characters are provided for billing messages. The operator now can add a specific interest rate or standard late charge to billings. All past due accounts may have interest charges calculated at the end of the month.

Patient Administration improvements include modifications to support a new "mixed mode accounting" feature allowing the user to define any specific account as open item, balance forward, or third party billing. Interest charges may or may not be specified for each account. The "escrow credit" balance field has been modified to reflect true "account balance". In addition, where the patient account number is not known, the operator may now enter the last name instead and the system can locate the account, using a SOUNDEX search method.

In Daily Processing, last name routine may be used instead of account numbers if desired. The limit of eight charges per invoice has been eliminated, and it is possible to set up a treatment case on one day but add related charges later. Each specific invoice can be assigned as a cash transaction, or as a time billing with several types of increments. All reorganization of indices now occurs only at the end of the day, so that daily interactive processes are sped up.

Primary and secondary diagnoses are now supported in the Medical version, as are primary and secondary insurance company assignments to a treatment expense. All charges on a specific treatment will be printed on the single insurance form, even if charges occurred on different days.

The Master Report section now includes inter-office reports and supports form feed printers. Pausing and Quitting are now permitted on most reports.

T/MAKER II
Version 2.5.1

This version introduces a number of enhancements. The print function now translates a given character in the working file into a series of characters or control codes to be sent to the printer when printing a file. Characters now may be entered with the eighth bit on; thus additional character sets can be created for a variety of purposes. A patch for Panasonic users has been added.

This version contains a new function, called BAR. It encompasses
a large number of commands, allowing rows and columns to be barred, so that bar charts can be created. Commands are divided into groups which determine the values to be barred, the presentation of charts, and the disposition of the charts. The width, spacing between bars and characters used to form the bars can be regulated, as can the beginning and end points of the bars on the charts. The lines on the bar charts can be numbered and the bars can be named. The bar chart files can be manipulated in various ways – combined, incorporated into working files, interspersed, etc. In addition, the bars can be suppressed or displayed as desired by the user. The BAR function is intended as an additional graphic tool to improve the readability of tables produced using T/MAKER II.

Bugs

BDS C Compiler
Version 1.45

A bug in the CLIB.COM program does not allow the user to open a CRL file for processing if a disk designator is specified as part of the filename and that filename is more than six characters long. This problem has been repaired in the same version, but some users will have CLIB.COMs with this fault; they can receive new CLIB.COMs.

In the NOBOOT document on the second page in the listing of patches to C.CCC, the hexadecimal byte at location 013A under the NOBOOT column header should read 39 instead of 19. This also has been repaired within version 1.45.

IBM/CPM

No version of this product runs with the CCS CPM-80; it also doesn't run with Lifeboat's CP/M-80 (version 2.25) for the TRS-80 Model II, or on the Hewlett-Packard 125.

Operating Systems

Description	Version
These operating systems are available from Lifeboat Associates, except where otherwise mentioned.	
CP/M-80 for:	
Apple II w/Microsoft BASIC	2.20B
Datapoint 1550/2150 DD/SS	2.21
Datapoint 1550/2150 DD/DS	2.21
Datapoint 1550/2150 DD/SS w/CYN	2.21
Datapoint 1550/2150 DD/DS w/CYN	2.21
Durango F-85	2.23
Heath H8 w/H17 Disk	1.43
Heath/Zenith H89	2.2
iCOM 3812	1.42
iCOM 3712 w/Altair Console	1.42
iCOM 3712 w/IMSAI Console	1.42
iCOM Microfloppy (# 2411)	1.41
iCOM 4511/Pertec D3000 Hard Disk	2.22
Intel MDS Single Density	1.4
Intel MDS Single Density	2.2
Intel MDS 800/230 Double Density	2.2
MITS Altair FD400, 510, 3202 Disk	1.41
MITS Altair FD400, 510, 3202 Disk	2.2
Micropolis Mod I - All Consoles	1.411
Micropolis Mod II - All Consoles	1.411
Micropolis Mod I	2.20B
Micropolis Mod II	2.20B
Compal Micropolis Mod II	1.4
Exidy Sorcerer Micropolis Mod I	1.42
Exidy Sorcerer Micropolis Mod II	1.42
Vector MZ Micropolis Mod II	1.411
Versatile 3B Micropolis Mod I	1.411
Versatile 4 Micropolis Mod II	1.411
Horizon North Star SD	1.41
Mostek MDX STD Bus	2.2
Ohio Scientific C3	2.24
Ohio Scientific C3-B/74	2.24B
Ohio Scientific C3-C'(Prime)/36	2.24B
Ohio Scientific C3-D/10	2.24A
Ohio Scientific C3-C	2.24A
Sol North Star SD	1.41
North Star SD IMSAI SIO Console	1.41
North Star SD MITS SIO Console	1.41
North Star SD	2.23A
North Star DD	1.45
North Star DD/QD	2.23A
Processor Technology Helios II	1.41
by Lifeboat/TRS-80 5 ¼"(Mod I)	1.41
by Lifeboat/TRS-80 Mod II	2.25C
by Cybernetics/TRS-80 Mod II	2.25

Hard Disk Modules

Description	Version
Corvus Module	2.1
APPLE-Corvus Module	2.1A
KONAN Phoenix Drive	1.8
Micropolis Microdisk	1.92
Pertec D3000/iCOM 4511	1.6
Tarbell Module	1.5
OSI CD-74 for OSI C3-B	1.2
OSI CD-36 for OSI C3-C'	1.2
SA 1004 for OSI C3-D	1.1
SA 4008 for OSI C3-C	1.3

(continued next page)

VERSION LIST

February 8, 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.

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

S	Standard Version
M	Modified Version
P	Processor
MR	Memory Required

Product	S	M	P	MR	
ACCESS-80	1.0		8080/Z80	54K	
Accounts Payable/Cybernetics	3.1		Z80	64K	Needs RM/COBOL
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		8080	48K	Needs BASIC-80 4.51
Accounting Plus			8080/Z80	64K	
Accounts Receivable/Cybernetics	3.1		Z80	64K	Needs RM/COBOL
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
ALDS TRSDOS		3.40	8080	32K	Needs TRSDOS. TRSDOS Macro-80
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	1.0	8080		Needs CBASIC2
ASM/XITAN	3.11		Z80		
Automated Patient History	1.2		8080	48K	
BASIC Compiler	5.3	5.3	8080	48K	
BASIC-80 Interpreter	5.21	5.21	8080	40K	w/Vers. 4.51,5.21
BASIC Utility Disk	2.0	2.0	8080	48K	
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	4.5	8080	32K	
BDS C Compiler	1.45	1.45T	8080	32K	w/'C' book
Whitesmiths' C Compiler	2.0		8080	60K	
BSTMS	1.2	1.2	8080	24K	
BUG / uBUG Debuggers	2.03		Z80	24K	
CBASIC2 Compiler	2.08		8080	32K	w/CRUN(2,204P, & 238)
CBS Applications Builder	1.3		8080	48K	Needs no support language
CIS COBOL Compiler	4.4,1		8080	48K	
CIS COBOL Compact	3.46	3.46	8080	32K	
FORMS 1 CIS COBOL Form Generator	1.06	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	4.01	4.01	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	2.03		8080	48K	Needs 80x24 terminal
dBASE-II	**2.3A**		**8080**	**48K**	
dBASE-II Demo	**2.3A**		**8080**	**48K**	
Dental Management System 8000	8.7A		8080	48K	Needs CBASIC
Dental Management System 9000	1.07		8080	48K	Needs CBASIC
DESPOOL Print Spooler	2.1A		8080		
DISILOG Z80 Disassembler	4.0	4.0	Z80		Zilog mnemonics
DISTEL Z80/8080 Disassembler	4.0		8080/Z80		Intel mnemonics,TDL extensions
Documate/Plus	1.4		8080	36K	
EDIT Text Editor	2.06		Z80		
EDIT-80 Text Editor	2.02	2.02	8080		
FABS-I	2.6		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	3.43	3.43	8080	36K	
FPL 56K Vers.	2.6		8080	56K	
FPL 48K Vers.	2.6		8080	48K	
General Ledger/Cybernetics	1.3C		Z80	48K	Needs RM/COBOL
General Ledger/MC	1.0		8080/Z80	56K	Needs CP/M 2.2 or MP/M
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

VERSION LIST

Product	S	M	P	MR	
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	3.43	8080/Z80		
MAG/base1	2.0.1		8080	56K	Needs CBASIC, 2.06 or later & 180K/drive
MAG/base2	2.0.1		8080	56K	Needs CBASIC, 2.06 or later & 180K/drive
MAG/base3	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
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	**1.1**		**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	2.04		8080	48K	Needs BASIC-80, 5.03 or later
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		1.1	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	2.10	2.12	8080		
Mu SIMP / Mu MATH Package	2.10		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		8080	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)	**2.2**		**8080**	**56K**	**Z80/8080**
PLAN80 Demo	1.0				
PL/I-80	1.3		8080	48K	
PLINK I Linking Loader	3.28		Z80	24K	
PLINK-II Linking Loader	1.10A		Z80	48K	
PMATE	3.02		8080	32K	
POSTMASTER Mail List System	3.5	3.5	8080	48K	
Professional Time Acctg	3.11a		8080	48K	Needs CBASIC2

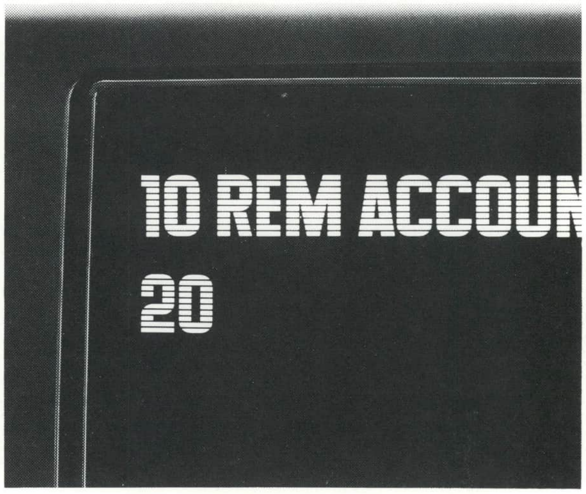
(continued next page)

VERSION LIST

Product	S	M	P	MR	
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	1.3C		8080	48K	w/Cybernetics CP/M 2
RAID	5.0.2	5.0.2	8080	28K	Modified for TRS-80 Model-I only!
RAID w/FPP	5.0.2	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	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
SMAL/80 Programming System	3.0		8080		For CP/M 1.x
Spellguard	2.0		8080/Z80	32K	Needs Word Processing Program
Standard Tax	1.0		8080	48K	Needs BASIC-80 4.51
STATPAK	1.2	1.2	8080		Needs BASIC-80 4.2 or above
STIFF UPPER LISP	**2.7**		**8080**	**48K**	
STRING BIT FORTRAN Routines	1.02	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	**2.5.1**		**8080**	**48K**	**Avail. for CDOS**
T/MAKER II DEMO	2.4		8080	48K	
TEX Text Formatter	2.1		8080	36K	
TEXTWRITER-III	3.6	3.6	8080	32K	
TINY C Interpreter	800102C		8080		
TINY C-II Compiler	800201		8080		
TRS-80 Customization Disk	1.3C		8080		
ULTRASORT II	4.1B		8080	48K	
Lifeboat Unlock	1.3		8080		Use w/BASIC-80 5.2
VISAM	2.3p		8080	48K	
Wiremaster			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		
XYBASIC Extended Compiler	2.0		8080		
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.

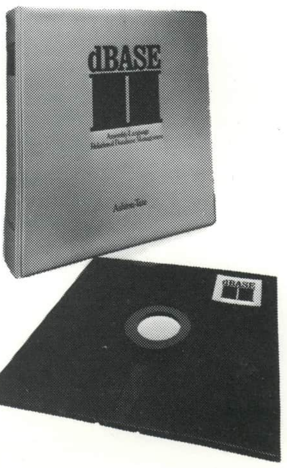
Call for our Dealer Plan and OEM run-time package prices, then take us up on our money-back guarantee. Send us your check and we'll send you a copy of dBASE II that you can exercise on your CP/M® system any way you want for 30 days.

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

Also available from Lifeboat Associates.

LIFELINES®

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

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

[redacted]