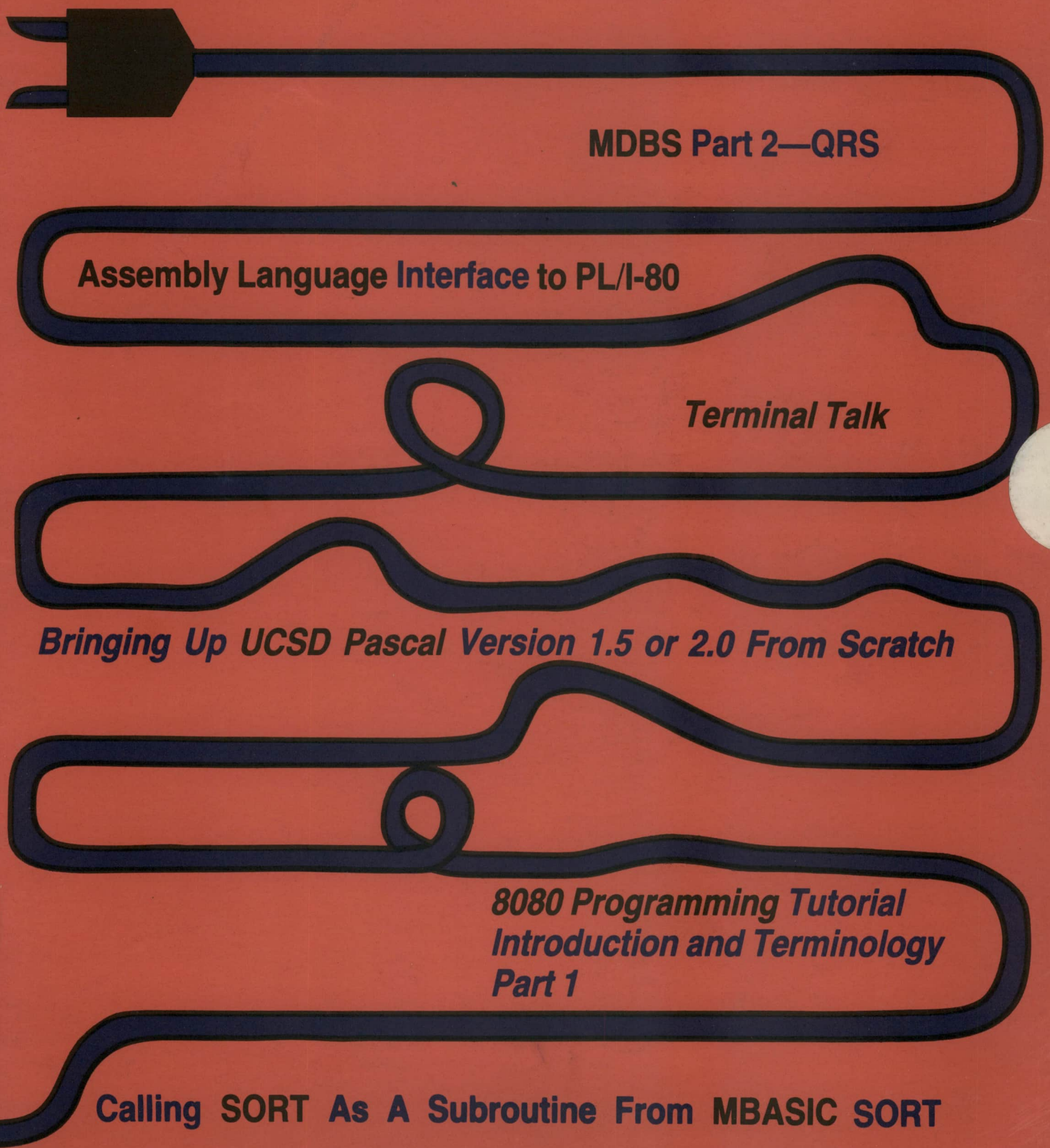
SEPTEMBER 1981

LIFELINES®

$2.50 VOLUME TWO NO. 4



MDBS Part 2—QRS

Assembly Language Interface to PL/I-80

Terminal Talk

Bringing Up UCSD Pascal Version 1.5 or 2.0 From Scratch

8080 Programming Tutorial
Introduction and Terminology
Part 1

Calling SORT As A Subroutine From MBASIC SORT

LIFELINES®

Editor-in-Chief: Harris Landgarten
Managing Editor: Jane V. Mellin
Production Assistant: K. Gartner
Administrative Assistant: Susan M. Sawyer

Volume II No.4 September

CONTENTS

Lifelines, Volume II, Number 4. Published monthly. The single copy price is $2.50 domestically, including the U.S., Canada, and Mexico. The single issue price for copies sent to all other countries is $3.60. A one year's (12 issues) subscription is priced at $18.00, when destined for the U.S., Canada, or Mexico, $40.00 when destined for any other country. All checks should be made payable to Lifelines Publishing Corporation, Foreign checks must be sent in U.S. dollars, drawn on a U.S. bank; checks, money orders, VISA, and MasterCard are acceptable. All orders must be prepaid. Lifelines is published by Lifelines Publishing Corporation, 1651 Third Avenue, New York, N.Y. 10028; telephone: 212-722-1700. Please send all correspondence to the Publisher at the above address. Postmaster send change of address to the above address. Application to mail at second class postage pending at New York, N.Y.

Editorial Comments

I'm sure that many of you have noticed the vast amount of inconsistency in the quality of software documentation among various authors. While much of the sub-standard material can be attributed to a rushed job by someone happy to have finally gotten the "last" bug, the larger problem may lie with a firmly established historical precedent which no longer applies.

Until five or six years ago, the scenario looked something like this: giant computer companies raced to build bigger and better machines for a customer base which felt lucky to obtain a machine in a matter of years. Software was of no concern to hardware manufacturers, since those lucky enough to have purchased one of their computers could easily have any necessary software generated by the in-house programming staff. The documentation supplied by the hardware maker was probably written by the engineering staff, and was clearly targeted at the highly trained experts employed by the company purchasing the machine. The application development tools, such as compilers and assemblers, were accompanied by reference manuals. The purpose of the reference manual was to clearly and tersely set forth the syntax of the language, making the assumption that those reading had extensive prior knowledge of the concepts involved, a valid assumption since anyone using a mainframe surely had access to some sort of training. Applications were either generated in-house to suit the needs of the user, or contracted for, from outside consultants. Either way, the documentation was only good enough to serve as a memory refresher to users who had the program authors available for extensive training and questioning.

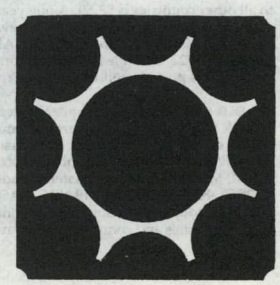
Unfortunately, some of these conventions have carried over into the microcomputer world of today. The scenario, however, has changed substantially. The would-be programmer in many cases has no formal training in the theories of computer languages, and the application user, while no less intelligent than his corporate counterpart, has no in-house expert to question. Therefore, it is necessary for documentation to meet new standards based upon a realistic assumption of prior knowledge. This need is especially transparent when applied to programming language manuals. Almost without exception, the old rule of the reference manual still dominates; undoubtably because compiler writers were indoctrinated to assume prior knowledge of the language. At the very least, all programming language documentation should include a list of tutorial books for both the novice and intermediate programmer. Caveat Emptor should certainly be the byword of the novice compiler buyer. If the product does not explicitly mention instructional material, it can be safely assumed that none is included in the purchase price. Wisdom would dictate that one wishing to learn to program in a new language should investigate locally available university courses. If you have used a programming language book which you found particularly helpful, please send me the name of the book along with the name of the language it explains (be specific in naming the language). *Lifelines* will try to compile a list of suggested reading for compiler users.

Documentation accompanying application software is generally better in terms of invalid assumptions. An application programmer, in many instances, will assume prior knowledge by the user of the topics the program deals with. While in some cases this is unavoidable, these assumptions should be held to a minimum. As many consultants will testify, small business bookkeepers are often-times uneducated in truly proper accounting procedures. Tutorial sections in accounting package manuals are therefore never out of place. When application documentation does fail, it is usually because of poor organization. You can't figure out what to do first, or no set procedure for repetitive operations is set forth clearly. The best manuals I've seen usually separate their tutorial function from their reference function. In this way, the new user can be "led down the path" in an illustrative manner, while the experienced user can refer to the indexed reference section for technical information.

Ultimately, it is probable that the most successful computer programs will attribute their widespread usage as much to outstanding documentation as to utility.

It seems that a strong new competitor is about to enter the battle for the affections of programmers. CB-80 was formally announced at a recent seminar given by Gordon Eubanks of Compiler Systems. Bill Burton was present at the seminar and informed me of the highlights. CB-80 is a three pass BASIC compiler which generates true native 8080 code, via the traditional REL file root. The language itself is a superset of Compiler System's popular CBASIC.. Among the additions to the language are: Nested IF THEN ELSE; Error Trapping; an INKEY function; Alpha-numeric Labels Lock and Unlock with MP/M2; and the ability to change individual bytes with a record. The authors claim that significant optimization will result in programs which are more compact and faster than either BASCOM or PL/I-80. Tentative release date is September 1st. No firm price has been announced. The library distribution license (necessary for selling compiled programs) calls for a $2000.00 per year payment which covers all code generated by a given compiler. Considering that Microsoft now has BRUN and Compiler Systems is licensing libraries to authors, the coming months should prove interesting. The possibilities seem endless. See you next month.

Harris Landgarten



The Pipeline

by Carl Warren

Take a look at graphics
Software and hardware designers are finding that there is more to the system world than alphanumeric displays and hardcopy output. At least one would gather so from visiting the Siggraph exposition held in Dallas, Texas this past August.

You could find on display everything from an Apple computer performing detailed trend analysis in a variety of colors to megabuck systems such as Genisco's Spacegraph doing simulated 3D displays.

What was significant though, was that the preponderance of equipment shown was for the business world. According to Ken Anderson, publisher of the Anderson newsletter—a newsletter to the graphics world, the need for graphics, specifically color graphics, has grown at a pace far faster than anyone really expected. Moreover, Anderson contends that much of its growth has been the result of end users realizing that computers, especially microcomputers, could greatly enhance their personal productivity. Therefore, as a consequence, they have demanded more from the micros and have pushed the industry for enhancements.

Another point that Anderson makes is that computer manufacturers such as Apple and Radio Shack, working in tandem with software developers like Personal Software, have shown that large amounts of data, that heretofore was presented as reams of printouts, could be reduced easily to a single display that conveyed easily understood information.

Others agree with Anderson. David Deans, V.P. of Marketing at Intelligent System Corp.(ISC), points out that for his line of color computers, ISC has developed a full line of CP/M compatible color graphics software. Furthermore, ISC is developing interfaces to interact with large mainframe graphics ware, and to support a variety of graphics hardware such as Houston Instrument's line of desktop plotters.

Not all graphics are relegated to specialized systems however. Virtually every manufacturer of printers is offering some kind of graphics capability. Epson, for example, is offering a ROM add-on for the popular MX-80 desktop printer. This ROM set gives you the capability of doing bit graphic hardcopy output from any system you can interface the unit to. In daisy wheel printers, Diablo offers an option board for the Model 630 that lets you easily slip in and out of a graphics mode for creating meaningful business graphics. But be aware, the Diablo printer can perform this function without the additional board, with the aid of user written software.

Not so surprising is that Diablo and Epson aren't the only printer manufacturers offering graphics capability. You might want to consider looking at Integral Data's Paper Tiger printers for bit graphics, or Oki Data. The latter has been working in tandem with Agent Computer Services to provide a wide range of graphics software for both the Apple and TRS-80 microcomputers.

Should graphics be your thing, you might want to explore the possibilities even further by adding a digitizing pad, available from a variety of manufacturers. The most popular is the Bit-Pad from Summagraphics.

By employing a digitizing pad, you can quickly develop working drawings that depict everything from the Mona Lisa to what a machinist might use.

As exciting as all these hardware items are, the real key boils down to software. Currently, very few software houses have developed useful packages that maximize the power available, thus opening the door to interesting and lucrative software designs.

Not to be left out of the rising need for high capability, graphics oriented software is Sorcim, the makers of SuperCalc. According to Richard Frank, the company's founder, Sorcim is already adding extras to the recently released electronic spread sheet. These extras permit the creation of graphics on a Houston instrument plotter from the base of data generated in SuperCalc.

Following the same trend is Ashton-Tate with dBase II. They expect sometime in the next ten to twelve months to offer an option that will let users create business type graphics from dBase II database items.

With the strong trend toward graphics, specifically business graphics, it appears that end-users will want more application-ware, but without a lot of work. Therefore, software designers now have the task of not only developing packages that perform a multiplicity of tasks, but are what is universally termed: user-friendly.

This function of being user-friendly implies that the user is extremely unsophisticated as far as computers go. This means that packages must come ready to run, or can be easily installed by employing such techniques as menu selection. Moreover, it may be more palatable to a user to select a single vendor and rely on them to provide all the software. This latter consideration may be the key to why Xerox, Hewlett Packard, and others may enjoy immediate and long lived success with their small desktop units.

Already Apple, Radio Shack, Vector Graphics, ISC, to name a few, are enjoying success by providing software that requires little or no so-called computer literacy on the part of the user. This, coupled with a wide ranging support base, has made the systems offered by these companies very acceptable to the business user.

Although graphics is important, you might still be in the mode where you are anxious to add more storage to your system. And if your system happens to be a Heath/Zenith Z-89, you might want to consider looking at Magnolia Microsystems' double density disk controller. This board gives you the ability to double the density on your 5.25 inch single sided floppies (the 90K units are boosted to 161K), and support 8-in. drives at the same time.

(continued next page)

The board comes with everything needed to quickly install it in the Z-89, (typically 45-min), is CP/M compatible, and permits the use of the Heath/Zenith controller in the system. For example, you can have the Heath board drive one or more drives and the Magnolia board handle the others. Magnolia permits communication with the Heath board, thus permitting you to upgrade your software library to the higher density.

Unlike the Heath board that requires the use of hard-sectored diskettes, the Magnolia controller is designed to support soft-sectoring.

Software installation is eased by the use of menus in a file called DDSETUP that guide you through all the necessary configurations. Be aware however, that Heath and Magnolia differ in device designations. Heath opted for their own method of defining disk drives, and printer drivers. Magnolia took a more standard approach; which by being standard may cause some confusion for Heath users.

An example of this confusion is the installation of WordStar 3.0. Most Heath users tend to request the TTY:driver on INSTALL. This won't work with the Magnolia version of CP/M without some redefining of the ports in the driver routines which Magnolia provides. A quick way around this is to choose the direct port driver option in INSTALL. When invoked WordStar will quicky configure itself to the desired port and provide the required handshaking to establish communication. When you choose this option, though, it dedicates the software to a specific system configuration, which in most cases is permissible, if not desirable.

Should you decide that you want to add the Magnolia board to your system, you might do well to buy your drives from them also. Magnolia suggests Qume Data Trak 8 drives and unfortunately only references those drives in the installation procedure, which can cause a problem. Even though other drives offer a compatible interface, the layout and option designations are most likely different, thus precluding ease in bringing the drive up.

If you're thinking about bringing up your own computerized bulletin board (CBBS), and aren't sure of the best way to go, you might want to contact Ward Christensen about his CBBS package. According to Ward, who can be reached via *Lifelines*, he is in the process of updating his CBBS package. The goal? To make it more user friendly, of course. Ward says that the more people use it the more they want. Furthermore, Ward says that users have been very vocal regarding what they liked and disliked about his system and as a result many improvements have been made.

Currently, there are various methods of putting a system on-line, but the most popular appears to use CP/M and possibly Ward's package. One approach that is coming into vogue, and touted by Microperipheral's president Mike Darland, is to tie hundreds of micros together nationwide and create a micro electronic switching system. Darland is still in the process of developing this idea, and promises to be offering some unique opportunities for micro owners, plus some exciting deals on his modem.

Databases are important for a number of reasons, but if you're in an information intensive business, like a writer is, you currently have a number of choices of software from which to choose to use on your dedicated system.

However, you might want to extend your capabilities and share your knowledge with others by employing the facilities of one of the timesharing systems such as MicroNet. By employing the services of such a system, you can extend your storage space, have interactive dialogue with other users, and provide a unique service to other micro owners. Compuserve's John Meier envisions users making MicroNet and other similar services an adjunct to their smaller systems.

Supporting this effort, Compuserve has already set up dedicated processors to improve system throughput, and is developing software. This latter offering will make it possible for you to enter totally unrelated information into a database and retrieve it in a manner suitable to you.

Once implemented, data stored on MicroNet can then be downloaded to your local system and be massaged by a data handler like dBase II for example. Currently, a system similar to this is being employed by one California businessman to handle large parts inventories he's responsible for. The software he uses consists of a datahandling system written in FORTRAN on MicroNet with a data structure interface to dBase II on his end. His purpose is two-fold: take advantage of extra storage possible on MicroNet, and to make available his inventory status and needs to his other office in Denver. Interestingly, his needs have already grown, and he is looking for ways of using the data to develop management reports and maybe a way of having graphics representation of the data.

A late news flash—
The IBM Personal Computer is here! And it is called the IBM Personal Computer. Based on the 8088, it is priced from $1565 to $6300. The basic configuration comes with a keyboard module, uses an audio tape cassette and plugs into a TV. Color graphics and BASIC are optional. An expanded system with color graphics, two 5 1/4" floppy drives of 160K each, an 80-CPS dot matrix printer, and 64K of memory is priced at $4500. It uses the Advanced Disk Operating System developed by IBM and Microsoft. IBM is currently working with Digital Research and SofTech for CP/M 86 and UCSD Pascal.

Hear Ye! Hear Ye!

A late news bulletin has just come in: Pascal MT + and the Microsoft Compilers are now up and running on the Apple. (The CP/M may need to be updated or patched—if your CP/M signs on as version 2.20B, it has the required fixes.)

Bringing-up UCSD Pascal Version 1.5 or 2.0 From Scratch

by Kelly Smith

My major project recently has been getting UCSD Pascal up and running on my 8080 system (an IMSAI with iCOM floppies and 56 kilobytes of memory). This has involved a certain amount of agony and frustration, most of which could have been avoided. If you are one of the lucky ones that managed to get copies of UCSD Pascal *prior* to the revoking of *all* license agreements that had been placed with UCSD (if you got all 14 diskettes, you have the source code!), and you *never* got it *running* . . . this is for *you*! Note: Of course you could always *buy* it from SofTech . . . they would be more than happy to sell you software .

There are a reasonable number of people out there going through the same process, and I thought that since I have (inadvertently) become an expert on all of the mistakes one can make bringing up UCSD Pascal, it might be helpful if I described the process, and perhaps save you some of the agony.

NOTE: This article applies to either version 1.5 or 2.0 (where references to 2.0 will be closed in "()"). The general assumption is, that you have an 8080 or Z80 system which has two floppies, at least 48 kilo-bytes of memory, and the Digital Research CP/M operating system, as well as the UCSD Pascal diskettes and the manual.

Getting Off the Ground

It's actually fairly easy to get UCSD Pascal up and running, but a lot harder to read enough of the manual to figure out how to do it. One secret is that the manual is best read from back to front! I will give page references to the manual as I go along, since it's index is nearly useless. On pages 273-275 (v-vii) you will find a reasonably concise description of how to get things off the ground. Let me paraphrase it: You start by getting CP/M up and running with existing CP/M disks, and then look among the various UCSD Pascal diskettes . . . One of them is called "CP/M". It is a diskette readable by your system, and on it is a file called PASCAL. COM. You put this diskette in the "B" drive, and PIP it to your "A" drive that has your CP/M on it. Note that although the diskette from UCSD is called "CP/M", it does not have the whole CP/M system on it, just a few UCSD utilities, but it is in CP/M diskette format. At this stage you should ignore all the stuff about creating a "bootstrap" on pages 274-275 (vi-vii); that can come later.

Now you are ready to get started. There should be a diskette among the UCSD diskettes called PASCAL8: (2.0 Volume U001A. B:). This is the MAIN system diskette. Locate it, and get a "scratch" (formatted and empty) diskette. Start "booting" by executing the file PASCAL. COM on your CP/M system diskette in drive "A" (just type PASCAL followed by carriage return). It will prompt (misspelled) by asking you to insert your "PASAL" diskette in drive "A". Remove your CP/M system diskette, and put in the PASCAL8: diskette as well as the "scratch" diskette in drive "B". Now hit carriage return, and after a few seconds, your console should display "Loading . . .", and after 10 to 20 seconds a message will appear "Welcoming" your diskette (not you) to the system. It will also tell you that today's date is sometime in 1978 (1979), but don't worry, we'll be fixing that soon.

Using the UCSD Pascal System

So now UCSD Pascal has signed-on . . . what do we do now ?!? Let's fix that date too today! This isn't just for appearance: every time the system creates a file on a diskette it puts on the date of its creation, and that helps you keep track of *when* you made that file. First I will give a summary of how you type system commands . . . you will notice your cursor sitting at the end of a long prompt line, which tells you "Command: E(dit, R(un, F(ile, C(omp, B(lah, B(lah, B(lah, . . . ". Each of these describes what happens when you type one character, either an "E" or an "R" or an "F", etc. . . . DON'T TYPE ANYTHING YET!!!

"E" calls the system screen editor, "R" runs a particular compiled program, etc. . . . The letter we want is "F". It calls a program called FILER, which is for copying, deleting, and generally "messing around" with files. It has the particular property that when you type "F", it reads in the FILER, then comes up with a whole new prompt line. In effect, this is a system with two levels, and with each command being just one character. "F" is the command that "pops" you from the outer command level «"Command:"» into the inner level «"Filer:"». You "pop" back out by typing "Q" (for QUIT) while in the FILER level.

Messing Around with Files

Alright, let's type "F". With luck, the FILER prompt will come-up shortly, offering you a new range of "goodies". They include:

V Lists names of your disks which are available.

Z Zeros out the directory of a diskette.

B Scans a diskette for "bad blocks".

L Does directory listings so you can see what files are on a diskette.

D Changes the date.

T Copies (Transfers) files.

C Changes the name of a file.

R Removes (deletes) a file.

Now we are ready to change the date . . . we start by typing "D", and the program should display the "old" date and ask you for a "new" one.

(continued next page)

Just type it in the same format as the one it typed out, and then hit carriage return. You should get a response telling you that the date has been changed, and "what to", and then you will find yourself back at the FILER prompt line again. (Note that the one line FILER menu is too short to list all the FILER commands available to you. If you wish to see what additional commands exist and what they are, type in a "?"; an "in depth" discussion of the FILER commands are listed in pages 7-30 [7-30] of the USCD manual.)

So now you have successfully executed a FILER command. Now the next thing you want to do is make a backup copy of your main system diskette! (I can't stress this enough, if you are a "newcomer" to UCSD Pascal). If we are going to copy a whole diskette, we need to get the system to recognize that your "scratch" diskette is really installed in your system. Start by typing "V". This is the FILER command that lists the "volumes" (disks and other devices) which are "on-line". The list that should appear on your console will show a series of numbered devices. CONSOLE and SYSTERM are your screen and keyboard. PASCAL8: (U001A.B) is your main system disk, and PRINTER is a listing device. But notice that it doesn't list anything corresponding to your other ("scratch") diskette . . . that's because it doesn't know it exists yet.

Introducing Your Diskette to UCSD Pascal

To inform UCSD Pascal of the existence of your "scratch" diskette, type "Z". This "zeros out" the directory of a diskette. Of course, it now asks you WHICH disk. It may strike you as absurd that you are to tell it which diskette when it doesn't know the name of the new diskette, but you aren't going to tell it a name . . . you will not type the name of the diskette, but its device number. Recall that your main system PASCAL8: (U001A.B) diskette was device number 4, the other is always number 5. From "Z" there follows a series of prompts asking for various things. Refer to the Z(ero writeup on pages 28-29 (29-30). I always say my disks have 494 sectors, and it seems to be happy with that. You will have to give the new volume a name. Choose one which is short (eight characters) enough to be easily typed, memorable, but not so short that you are likely to type it by mistake. Let's say that it is SYSTEM: (volume names are always followed by colons, otherwise the system thinks they are file names). When you have finished doing the "Z" command and are back to the FILER level, do a "V" command again to see if the system now knows that SYSTEM: is there.

At this stage you may want to give an "L" command at the FILER LEVEL. This LISTS the names of all files on a given volume. You have to tell it the volume name, which will be either PASCAL8: (V001A. B) or SYSTEM: . You can look at the list of names of files on PASCAL8: to familiarize yourself, and on SYSTEM: to make sure there aren't any (still an "empty diskette").

Now type "B" to do a B(ad blocks scan of SYSTEM: to make sure that there are no bad blocks there (Reference page 26 (25) of the manual). Now we are ready to copy all of the stuff on PASCAL8: to "scratch" diskette SYSTEM: . This is done by typing "T" while in the FILER. It will ask for the name of the file you want to copy (Transfer). Now you may enter all of the files on your system diskette, one by one, but that would be tedious (also you don't know them yet).

What you can use instead is the "wild-card" character, which is an equal sign ("="). If you ask it to T(ransfer the "file" named "*:=" to SYSTEM: = (i.e. *:=,SYSTEM:=«cr») it will copy EVERY file from your default volume (#4, named PASCAL8: or you will see the name of each file appear on your console as they are copied from one volume to the other. When the system returns to the FILER level, you should try an "L" command to check that SYSTEM: (or equivalently, #5:) has all the files that were on PASCAL8: (or U001A: .B).

Two Name Changes Required

You are now almost ready to start writing programs, but first set PASCAL8: aside (our "backup" diskette) and go through the "booting" process again (again, a "scratch" diskette in drive "B") with our new SYSTEM: diskette in drive "A" as the main system diskette. Note also, that you may want to do FILER commands "Z" and "B" on the "scratch" diskette, so as to make it available as a place to T(ransfer) files too . . . Now we have our "backup" PASCAL8: out of harm's way and are using the copy SYSTEM:.

At this point the temptation to type "E" is getting strong. Surely we want to get started typing in a program! Don't yet!!! The problem is that when you are at the COMMAND level and type "E", the system goes out to your diskette and looks for a file called SYSTEM. EDITOR, and starts running the code in that file . . . the catch here, is that that's the Screen-Oriented Editor, and (no doubt) does not know how your console random cursor addressing works! It's a nifty editor, but you can't run it until you go through a (rather elaborate) process of writing a Pascal routine called GOTOXY and running two programs called BINDER and SETUP. This is not only tricky, but is very system dependent, so we will leave it aside. Right now, if you type "E" at the COMMAND level, all that happens is that you "crash" the system!

There is another editor on your disk however, called YALOE (Yet Another Line Oriented Editor [should have been called "Son-of-TECO"]), and we will use that. You can run it by typing in "X" (for eXecute) at the COMMAND level, and then YALOE«cr». Now, using the "C" command (at the FILER level), CHANGE the file name SYSTEM. EDITOR to SCREEN. EDITOR (pages 18-19 in the 1.5 and 2.0 manuals). Then change YALOE. CODE to SYSTEM. EDITOR. You might at this point want to R(emove (pages 20-21 in 1.5 and 2.0 manual) the extra copy YALOE. CODE (NOTE . . . this is the command that erase files, and should be used with great care!). Now you can edit using YALOE by getting to the COMMAND level and typing "E". There is also one other change that you should consider. On the system disk is a file named SYSTEM. MICRO, the main system interpreter, which runs ALL code. But there is also a file called 8080T. CODE, which is another copy of the interpreter (a longer

one). The difference is that it has the capability of computing the transcendental functions (like LOG, SIN, etc.) while this is left out of the SYSTEM. MICRO file. If you are going to be using these scientific functions, you must C(hange SYSTEM.MICRO to (say something like) 8080U.MICRO, and then 8080T.MICRO to SYSTEM.MICRO. Then it would be wise to "re-boot" from CP/M to make sure that the new interpreter is the one resident in memory.

At Last!!!

Now we're finally ('bout time!) ready to edit a program and run it. Get to the COMMAND level (type "Q" if in the FILER), and then type "E". A YALOE prompt should appear, and it will tell you that you have no workfile. The editors always "work" from a workfile called SYSTEM. WRK. TEST, your workfile. At the FILER level there are four commands to deal with workfiles:

N(ew — Throws away the present workfile so that you can start with a "clean slate".

G(et — Tells the system which file you want to use as the workfile, also throwing away any pre-existing workfile.

W(hat — Asks for the name of the workfile (although after you edited it, appears to forget the name, which is not a tragedy).

S(ave — Writes the workfile to a file of your choice (file name), destroying any old copy of that file in the process.

You won't have to use any of these yet (reference pages 13-15 [13-15]), for now you might want to know some YALOE commands so you can type in your first program. Note: After you get out of the editor, the edited stuff is kept in SYSTEM. WRK. TEXT, which means you don't have to overwrite the original file. YALOE is written up on pages 59-69 (63-73) of the manual. Here are a few of the commands:

B — Go back to the Beginning of the file.

nA — Advance "n" number of lines.

nL — List "n" number of lines (but don't move the pointer!)

nF — Find the n-th occurrence of a string.

nD — Delete the next "n" number of characters.

I — Insert.

QU — Quit and Update SYSTEM. WRK. TEXT

QE — Quit and Exit (don't update SYSTEM. WRK. TEXT).

V — Verify (display) the current line.

If a command doesn't have a value of "n", it takes it to be one. Negative values are allowed. Each command is terminated by hitting the Escape (ESC) key, which will then display a "$" character. Also, commands can be strung together in a line, and they will not be executed until you type an extra Escape("$$"). For example, to issue a command which finds the string "Kelly", deletes it, then inserts "Winsor" at that point, type (after the "$" prompt):

FKelly$-3D$IWinsor$V$$

Alright, you are ready to edit a file. Here's a program to type in (the "$" really is a dollar-sign):

```
(*$S+*)
PROGRAM TEST;
BEGIN
   WRITELN('Winsor Brown is a
Pascal Phreaque');
END.
```

Use "I" to type it in (you can type in a reasonable number of lines on one Insert command). At the end of the inserted Pascal statements, type Escape twice, and use -10L$$ to display what you have typed. Then use QU$$ to write the result out to SYSTEM. WRK. TEXT.

Here we Go!

To compile the Pascal statements you have typed in "P-Code" (pseudo machine code), make sure you are back at the COMMAND level, then type "C". You will get compiler messages, and it will link (if necessary), and return to the COMMAND level. Now type "R" to run the compiled code, and you will get "Winsor Brown is a Pascal Phreaque" displayed on your console . . . You have done it!

You may have noticed the mysterious comment line "(*$S+*)" at the beginning of the sample program. It tells the compiler that you don't have enough memory to do everything in memory, so it should write out stuff to disk. It isn't necessary (or so I am told) if you have 56 kilo-bytes of memory, but it sure is if you only have 48 kilo-bytes (without it, you get stack overflow and crash the system). Compiler directives look like this:

(*$S+,G+*)

. . . and are the first line in the program, as you need them. They are fully described on pages 81-87 (77-83) of the UCSD Pascal manual. You will also want to read about "Differences Between UCSD PASCAL and Standard PASCAL" on pages 135-158 (133-158). You will rapidly run out of space on your main system diskette, and may have to make a copy, leaving out some of the non-essential stuff like the Calculator and the BASIC compiler. In addition, you will also need to learn to use "B(ad" and "K(runch" on pages 26-27 (25-27) in the manual.

A Note On XASM-51

Assembling the test file (TEST51.ASM) in Version 1.07 should flag a "B" error on Bit 8, EQUATE. The format is intentionally incorrect, to show the error:

```
B000  BIT8  EQU  0F1H.0
```

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 your label. This will help the Lifelines Circulation Department to expedite your request.

New Address:

NAME

COMPANY

STREET ADDRESS

CITY STATE

ZIP CODE

Old Address:

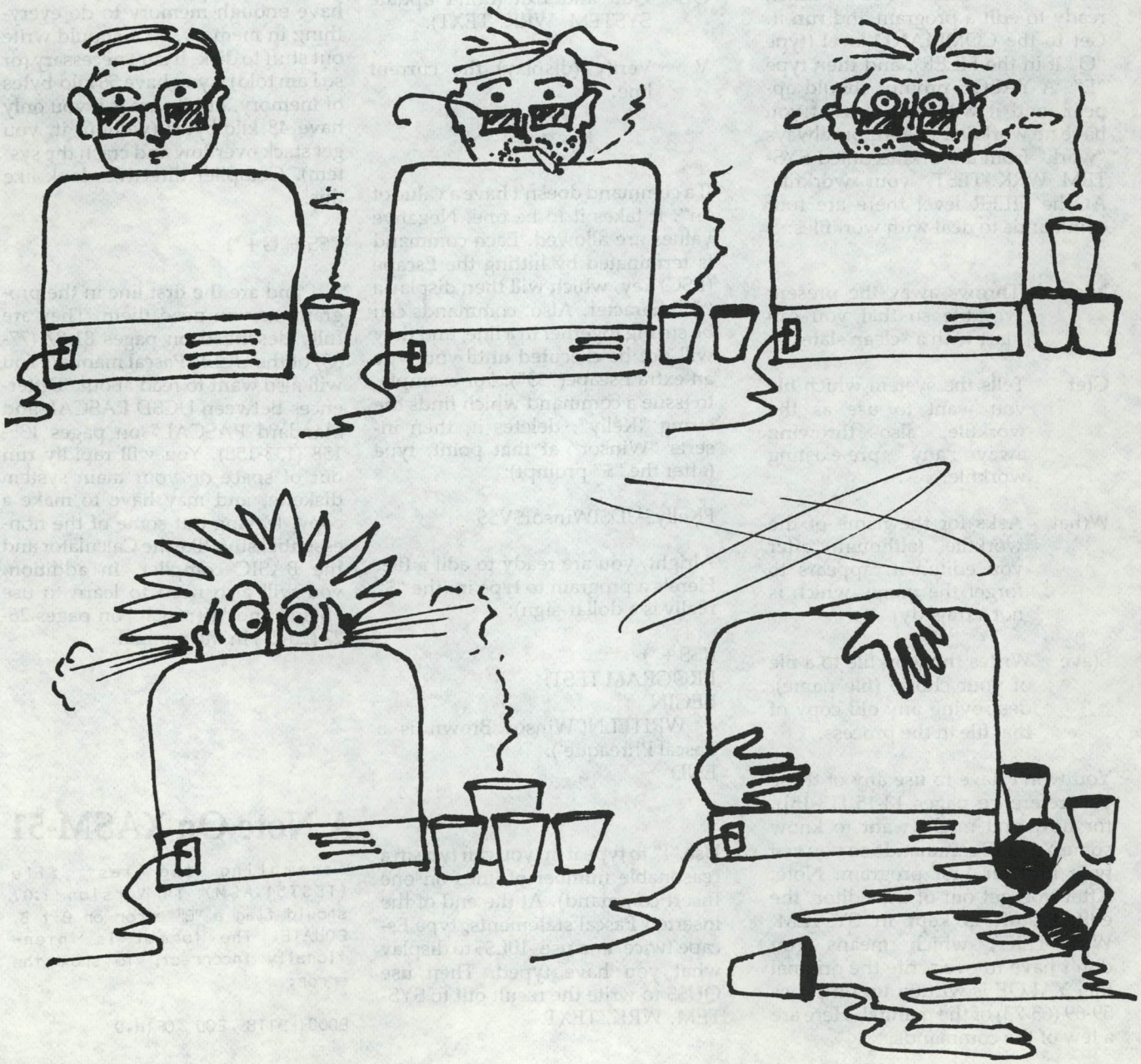
NAME

COMPANY

STREET ADDRESS

CITY STATE

ZIP CODE



MDBS Part 2 -QRS- The Query System/Report Writer

by Harris Landgarten

In Part One of this series, I pointed out some serious deficiencies in MDBS which would affect the usefulness of the system in a production environment. The MDBS authors, aware of these shortcomings, provide several database utilities and additions as options. The most popular optional utility, MDBS.QRS, is the query system and report writer. This stand-alone program, supplied as QRS.COM, is invoked by typing QRS and then answering some questions relating to database name and password (the question answering can be avoided by constructing a startup file). The QRS program consists of two unrelated parts: a non-procedural, English-like query language for database interrogation with a somewhat restrictive report writer facility; and an interactive environment for testing DML commands.

The query language part of this program is geared towards allowing the non-programmer to look at and change data in a MDBS database by writing simple query commands. Database queries are invoked by the commands, LIST, WRITE, STAT, and CHANGE. The generic format of a command is:

‹COMMAND› ‹FIND CLAUSE› ‹CONDITIONAL CLAUSE› ‹PATH CLAUSE›

The FIND CLAUSE consists of a list of data item names separated by commas. Optionally, the user may specify expressions involving data items in place of the data items themselves. The optional CONDITIONAL CLAUSE allows selective use of data based on the values of user selected fields. The PATH CLAUSE specifies the path of sets to be transversed in search of the required data and is probably the biggest single weakness in the package, since it requires that the non-programmer have an intimate knowledge of the structure of the database. This weakness can be overcome in the case of scheduled queries by predefining reports using the macro definition facility (more about that later).

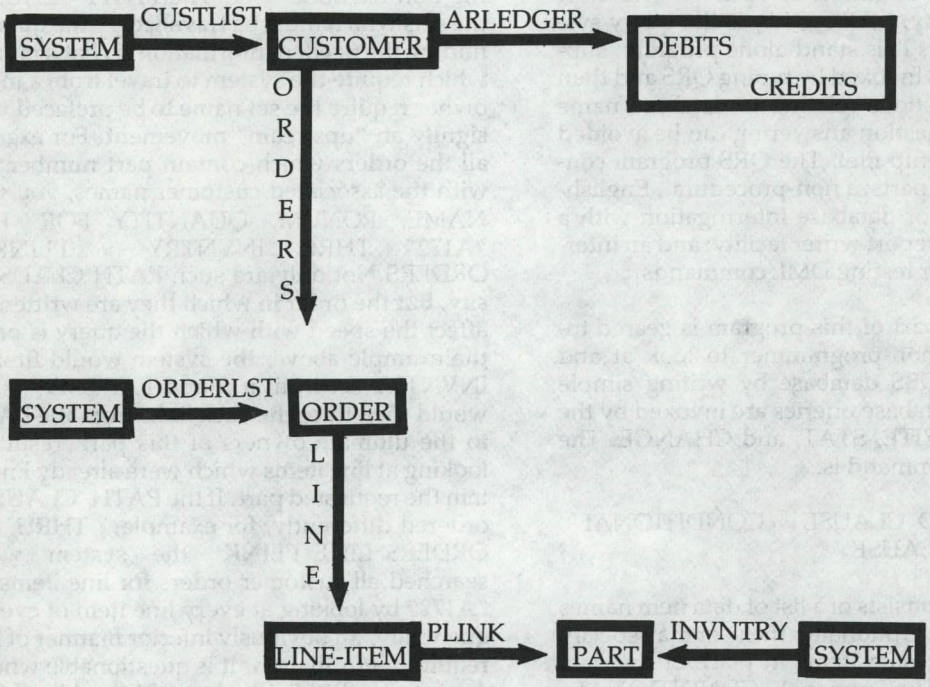
If this is confusing, perhaps an example using the order entry/accounts receivable database defined in Part One of this series will help (See Figure 1 and Listings 1). The LIST command simply prints the requested data in columns on the CRT screen. Suppose we wanted a list of all customers' names with their account numbers. This query would be LIST NAME, ACCTNO THRU CUST LIST. Since MDBS does not keep track of the implied decimal in fixed decimal numbers, it is the responsibility of the query writer to divide such numbers by the proper scale factor. In our example, all decimal numbers must be divided by 100, so a query to list part numbers, descriptions and prices from inventory must be written as LIST PARTNO, DESC, PRICE1/100THRU INVNTRY. Using conditionals will allow you to find all customers who owe money with the query LIST NAME, BALANCE/100 FOR BALANCE › 0 THRU CUSTLIST. Conditional tests can be constructed using all standard relations and boolean expressions allowing maximum flexibility. More complex queries require more complex PATH CLAUSES, which may be beyond the non-technical user. The PATH CLAUSE specifies the sets which must be travelled by the query system in finding the needed information. What's more, set paths which require the system to travel from a member to an owner require the set name to be prefaced with a "»" to signify an "upstream" movement. For example, to list all the orders which contain part number A172 along with the associated customer names, you would LIST NAME, PONUM, QUANTITY FOR PARTNO= "A172" THRU INVNTRY › PLINK ›LINE› ORDERS.Not only are such PATH CLAUSES a necessity, but the order in which they are written can greatly affect the speed with which the query is processed. In the example above, the system would first search the INVNTRY set until it found PARTNO = "A172". It would then trace the back links from the PART "A172" to the ultimate owners of this part, resulting in only looking at line items which were already known to contain the requested part. If the PATH CLAUSE had been ordered differently, for example, THRU CUSTLIST ORDERS LINE PLINK, the system would have searched all customer orders for line items containing "A172" by looking at every line item of every order sequentially, an obviously inferior manner of locating the required information. It is questionable whether or not an untrained individual would be able to format such queries with only the aid of a database diagram as suggested by the manual.

The WRITE command works similarly to LIST except that all column headings are suppressed and terms are written one per line, producing a "flat file" which can be stored to disk for later input to other software packages. The STATS command causes the min, max, average, standard deviation, variance, sum, and number of observations of the numeric terms in the report to be computed. (For non-numeric terms, only the number of observations is calculated.) For example, STATS TOTAL THRU ORDERLIST will print all the mentioned statistics for the total amounts of customer orders. A similar type of statistics reporting feature can be applied at any point in a report by inserting control breaks, using the BY clause. To obtain a report of the cumulative quantities of all parts on orders, you would type, LIST PARTNO,QUANTITY BY PARTNO THRU INVNTRY> PLINK, which would cause a numeric summary of QUANTITY every time PARTNO changed.

The final command is CHANGE which can work either interactively or automatically. The command CHANGE QUANTITY FOR PARTNO = "B1222" THRU INVNTRY > PLINK would cause the system to pause after displaying each line item containing part "B1222" and wait for input. If the operator responded with an

amount, that amount would be recorded as the new quantity; a carriage return would signify no change. The system then would proceed to the next occurrence. This might be used to selectively alter the amounts which would be ordered of a part in short supply. The automatic mode of CHANGE is used to implement global alterations. The command CHANGE PRICE1 PRICE1 * .15 FOR PRICE1< 3.00 THRU INVNTRY would automatically cause a 15% increase in all PRICE1s less than $3.00. Only one data-item type can be changed at a time using this command. Furthermore, the item changed cannot be the sort key of any sets specified in the THRU clause.

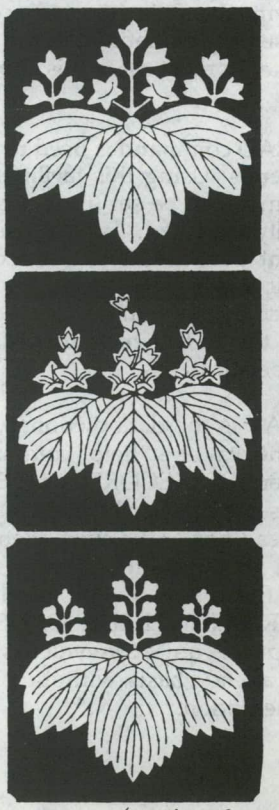
Figure 1 Diagram of Sample DB



Listing 1 Data description for Sample DB

```
0100    FILES    B:SAMPLE   .DB      1   512
0110    DRIVE    1          1000
0120    PASSWORDS
0130             ANYONE     255   255 PASSWORD

0140    RECORD   CUSTOMER
0150    ITEM     NAME       CHAR    30
0160    ITEM     ACCTNO     BIN     2
0170    ITEM     ADDRESS    CHAR    25
0180    ITEM     CITY       CHAR    15
0190    ITEM     STATE      CHAR    2
0200    ITEM     ZIP        CHAR    9
0210    ITEM     PHONE      CHAR    12
0220    ITEM     BALANCE    IDEC    10

0230    RECORD   DEBIT
0240    ITEM     DATE       IDEC    6
0250    ITEM     DESC       CHAR    12
0260    ITEM     AMT        IDEC    10
0270    ITEM     INVNUM     IDEC    5
0280    ITEM     PDATE      IDEC    6

0290    RECORD   CREDIT
0300    ITEM     DATE       IDEC    6
0310    ITEM     AMT        IDEC    10
0320    ITEM     AMTLEFT    IDEC    10

0340    RECORD   ORDER
0350    ITEM     PONUM      CHAR    12
0360    ITEM     ACCTNO     BIN     2
0370    ITEM     ORDDATE    IDEC    6
0380    ITEM     SHIPNAME   CHAR    30
```

```
0390     ITEM     SHIPADD1  CHAR  30
0400     ITEM     SHIPADD2  CHAR  30
0410     ITEM     SHIPADD3  CHAR  30
0420     ITEM     SHIPDATE  IDEC   6
0430     ITEM     STATUS    CHAR   1
0440     ITEM     VIA       CHAR  10
0450     ITEM     INVNUM    CHAR   5
0460     ITEM     TOTAL     IDEC  10
0470     ITEM     COST      IDEC  10
0480     ITEM1    NLINES    BIN    1

0490     RECORD LITEM
0500     ITEM     QUANT     BIN    2
0510     ITEM     PRICE     IDEC   6

0520     RECORD PART
0530     ITEM     PARTNO    CHAR  10
0540     ITEM     DESC      CHAR  40
0550     ITEM     PRICE1    IDEC   6
0560     ITEM     PRICE2    IDEC   6
0570     ITEM     PRICE3    IDEC   6
0580     ITEM     INSTOCK   BIN    2
0590     ITEM     INPROC    BIN    2
0600     ITEM     ONORD     BIN    2
0610     ITEM     REORDER   BIN    2
0620     ITEM     COST      IDEC   6

0630     SET      CUSTLIST  AUTO  1:N
0640                                    SORTED ACCTNO
0650     OWNER    SYSTEM
0660     MEMBER   CUSTOMER

0670     SET      ARLEDGER  MAN   1:N
0680                                    SORTED DATE
0690     OWNER    CUSTOMER
0700     MEMBER   DEBIT
0710     MEMBER   CREDIT

0720     SET      ORDERS    MAN   1:N
0730                                    FIFO
0740     OWNER    CUSTOMER
0750     MEMBER   ORDER

0760     SET      LINE      MAN   1:N
0770                                    FIFO
0780     OWNER    ORDER
0790     MEMBER   LITEM

0800     SET      ORDER     AUTO  1:N
0810                                    SORTED ORDDATE
0820     OWNER    SYSTEM
0830     MEMBER   ORDER

0840     SET      INVNTRY   AUTO  1:N
0850                                    SORTED PARTNO
0860     OWNER    SYSTEM
0870     MEMBER   PART

0880     SET      PLINK     MAN   1:1
0890                                    IMMAT
0900     OWNER    LITEM
0910     MEMBER   PART

0920     SET      ORDERLST  AUTO  1:N
0930                                    SORTED ACCTNO
0940     OWNER    SYSTEM
0950     MEMBER   ORDER

0960     END
```



(continued next page)

FIVE types of "Utility Queries": SET, EDIT,READ, DISPLAY, and CALC, are supplied as user aids. The SET command is used to change the value of environmental parameters which are constants, and switches which define the operating environment. A listing of these parameters is in Table 1. The syntax is simply SET parameter = value. SET also may be used to define column headings which ordinarily default to the names of the data-items displayed in the report.

EDIT, which is one of the most useful features of this package, is used to define macros, synonyms and value labels. Macro definitions allow the user to substitute single word commands for text strings of arbitrary length. Once a macro is defined, it is stored in the database itself for future reference, and will remain active until it is either altered or removed. Among the many ways in which this facility can be used are defining often used or complex queries as macros, so that unskilled users can invoke reports via one word commands. Another use might be the assignment of awkward expressions to more meaningful synonyms. For example, the term BALANCE/100 could be defined as BAL or a repeatedly used multi-term path could be defined as PATH1. Once such synonyms are defined, they can be used in place of the terms they represent either directly in queries, or in other macro definitions. The EDIT command is also used to define label values for different data item types in the database. In many cases, single character data values are used to represent a limited range of choices that a variable may take such as using R, B, and G instead of RED, BLUE, and GREEN as the values of a data item representing color. In a report, however, it is desirable to substitute the more descriptive label for the compact value. EDIT allows for the automatic substitution of such value labels by facilitating the construction of synonym tables for the data items requiring such a resource on an item by item basis. Synonym tables are, however, constrained to fit on one database page as defined in the DDL description of the database. In most cases this should not present a problem.

The READ command is used to read a disk file containing queries. Using this facility, complex sequences of commands can be set up in a command file and automatically executed. Typical uses are initializing environment parameters to desired values and initiating periodic report sequences. The command file can be prepared by any standard text editor or the editing facilities of the DDL which are ordinarily used to prepare DDL specifications.

DISPLAY is the data dictionary utility. It functions in the same way as a "help" command, by displaying a list of either SET, RECORD, or ITEM types present in the database. It can also be used to display the current column headings or the current environmental parameter values. In our sample database, the command DISPLAY RECORDS would produce the output "CUSTOMER DEBIT CREDIT ORDER LITEM PART". This utility proves quite convenient when working with QRS on an *ad hoc* basis without proper written database documentation.

The CALC feature gives the user access to a simple on-screen calculator which can be used to solve arithmetic problems without leaving the QRS environment. The result of the last calculation is stored in the variable "#" and can be used in subsequent calculations or can be substituted for numeric values in ordinary queries.

TABLE 1

Parameter	Description	Default Value
CW	Console Width	80
CD	Console Depth	24
PW	Printer Width	120
PD	Printer Depth	60
PM	Printer Margin	0
LF	File name to which query results can be written	none
SP	Inter-Column Space	2
CP	Console Page Character	00H
PP	Printer Page Character	00H
OF	Output Format (first digit is number of digits preceding decimal point (max 8), second is number of digits after the decimal (max 9)	89
M1	One character wild card	"*"
MS	Multi character wild card	"$"
TL	Report Title	none

	Description if parameter is 1	Default
OPT1	I/0 echoed to printer	0
OPT2	Reports displayed on console	1
OPT3	Suppress printer output	1
OPT4	Write output to disk	0
OPT5	Echo commands read from input files	1
OPT6	Display output from WRITE command on console	0
OPT7	Print output from WRITE command on the printer	0
OPT10	Print all error response in Interactive DML queries	0
OPT11	Print all non-zero error response in Interactive DML queries	1
OPT12	Suppress column headings	0
OPT13	Suppress value lablels	1
OPT14	List one term per line in LIST	0
OPT21	Print number of observations	1
OPT22	Print max	1
OPT23	Print min	1
OPT24	Print sum	1
OPT25	Print mean	1
OPT26	Print variance	1
OPT27	Print standard deviation	1

The Interactive DML part of the QRS package allows the programmer to test DML commands in an interpretive environment. By merely typing DML commands, the results are instantly displayed, making this an important tool for debugging, testing and learning. The use of the Interactive DML is similar in concept to using a BASIC interpreter in direct mode. You type commands such as FMSK CUSTOMER, which stands for Find Member using Sort Key, and after typing in the prompted-for Key, the system returns the value of the error code. You may then type GETC for Get Current of run unit, and the system will display the found customer record. All the DML commands which are otherwise accessible only through applications programs may be executed in this fashion. With this tool, the programmer can poke around a database, looking at important values, checking relationships, and correcting potential problems, thus saving the hours of programming it would have taken just to construct test procedures or utilities which will only be used one or two times. In my opinion, the QRS package is worth having just for the Interactive DML facility alone.

In operation the QRS package performed flawlessly. The response time for producing reports is of course dependent on the complexity of both the query and the database, but I found that any report I tried ran to completion in under ten minutes. In any case, the speed with which this package produces information should exceed the equivalent function performed by a handwritten application program because of the extra buffer space inherent in the use of QRS. My only criticisms of the package, other than the path clause requirements mentioned earlier, are the severe limitiations of report formats. The package is only capable of printing rough looking columnar reports which are only suitable for situations where content is the major consideration. In many instances the only way to construct a variety of commonplace professional-looking reports is to use the WRITE command as a "front end" for other software. For example, there is no direct way to print invoices or statements with QRS despite the fact that all of the required information can be output. A top quality print format feature is an addition which should be seriously considered. Overall, considering the features of QRS, I think it would be a foolish mistake for any serious MDBS user to try to do without it. Next month, I will conclude this series with a description of the Recovery/Transaction Logging System, and the Dynamic Restructuring System.

Table II

Package or version name: MDBS.QRS Version 1.02 for PL/I-80 with Z80

Price: $300.00

Systems available for: CP/M, North Star DOS

Required supporting software: MDBS or HDBS and possibly a standard text editor

Memory Requirements: 52K minimum.

Disk capacity required: 160k minimum.

Utility programs provided: None

Record size & type limits: Types correpond to database definition as defined using MDBS. Records may be of any size up to 65k. Practical limit 4k. Database size is limited by your hardware and operating system.

Portability: Good portability between CP/M systems. Database can be selectively unloaded and written to disk as a flat ASCII file using the WRITE command.

User skill level required: Novice with some training in the structure of the database being queried. A basic understanding of the MDBS data manipulation language is necessary for use of the interactive DML feature. Can be used by a total novice if queries are set up and stored ahead of time.

Table III

Qualitative Factors

Documentation	
oranization for learning	4
organization for reference	4
readability	4
includes all needed information	4
Ease of use	
initial start up	5
conversion of external data	6
application implementation	3
operator use	6
Error recovery	
from input error	3
restart from interruption	1
from data media damage	1
Support	
for initial start up	4
for system improvement	4

(note that all error recovery depends upon proper back-up procedures. The RTL module is designed to aid in error recovery.)

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

(continued next page)

TABLE IV

Data Management Capabilties

A. Underlying Data Model:

1. **Data Types**— alphanumeric, numeric
2. **Relationships**— one to many, many to one, one to one, many to many

B. Functions Provided:

1.a. **Data dictionary maintenance**— automatically maintained internally by system.

b. **Data reorganization and conversion**— reorganization is not possible with module. Limited conversion facilities are provided by the CHANGE command which allows numeric processing on a global or local level. QRS also provides a value label substitution facility.

2.a **Data entry and editing**— editing is provided by the CHANGE command. Other data entry and editing can be accomplished by using the interactive DML commands in a similar manner to the one you would use in an application program. No direct facilities are provided for entry of new data.

b. **Report generation**— flexible content is available in a limited column format.

3.a **Data selection by predicate**— Excellent support of complex predicates using conditional clauses in QRS commands. Included all normally used relations and boolean expressions.

b. **Data joining & relating multiple data sets**— supports the powerful facilities provided by MDBS.

c. **Calculation on data**— automatic calculation of popular statistics on numeric data items either globally for a report or at user specified multiple break levels. Integral calculator provided for arithmetic calculations based on input data. Query commands support complex algebraic expressions.

4.a **Data independent application interface**— Database is fully independent.

OOPS!

On page 26 of the August *Lifelines* (Volume II, Number 3), the second and third sentences of the last paragraph in the lefthand column should read:

This means that if the routine is called with an argument of zero, the result is the sampled value of register R. If called with an argument of one, then the result is the sampled value of register R plus 256.

Coming Soon...

Next month we'll be hearing more from Mike Karas on Assembly Language and PL/I; Ward Christensen will continue his tutorial on 8080 Programming.

For future issues *Lifelines* is also planning a review of Pascal MT+, reviews of MAGSAM III and IV, Fabs, and MicroSeed. Reports on various sort programs, a wealth of CPMUG volumes, and a special series on language interfaces lie ahead.

We like to hear what you think of *Lifelines*, and we're eager to find out your opinions on the world of software as a whole. So keep those letters coming!

Attention Dealers!

If you are interested in selling *Lifelines*, let us know. We have a dealer package for you. Write to: Dealer Sales Department, Lifelines Publishing Corp., 1651 Third Ave., New York, N.Y. 10028. Or call (212) 722-1700 and ask for Susan Sawyer.

Terminal Talk

by Steve Patchen

INTRODUCTION

Contemporary software for small computers frequently tries to take advantage of CRT terminal features. These features are usually provided by terminal manufacturers to allow design of better user-oriented interactions with the computer. Unfortunately these manufacturers do not conform to the existing ANSI terminal standard or to any other standard. Terminal features, especially those relating to cursor control, therefore vary considerably. This makes it very difficult to write software for use on non-specific CRT terminals without giving up direct control of the terminal's cursor and high speed clear and re-write capability.

There have been two basic approaches to solving this problem. The first is to write a special version of the software for each terminal upon which it is intended to run. The second approach is to allow the user to create a table of information to be used by the program to define the terminal behavior. The first approach provides the most reliable programs because the terminal interaction can be tested by the program's authors. The second method has the potential for the widest use because it can be used by terminals not considered by software authors. This second approach runs into a lot of problems however. It is this second approach and its problems which this article will discuss.

There are two major systems in a terminal: the display and the keyboard. The two systems are independent enough to be considered separately. I will therefore put off discussion of the keyboard until a later article and concentrate upon the display in this article. The purpose of this article is to identify terminal display features which make terminal independent software difficult to write; and to identify the assumptions about terminals which software authors frequently make and which cause the software to be unusable at many installations. This article will be followed by one or more articles discussing the details of terminal installation of one or more software packages. A comparison of installation approaches will then be made along with some recommendations for both software authors and users.

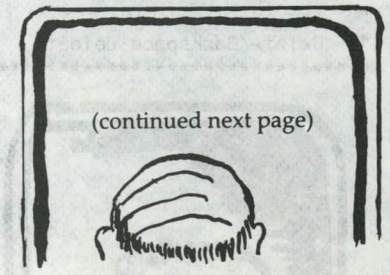
I welcome any feedback and contributions from readers and authors during the course of this series.

TERMINAL FEATURES

The features which most enhance the interface with a terminal user are the ability to clear the screen and re-write it or to write a new screen of text, and the ability to place the cursor at any location on the screen in order to write to that location or to accept keyboard entry from that location. Editing is enhanced by the ability of the terminal to insert blank lines on the screen and scroll the lower lines out of the way, or to delete a screen line and scroll the lower lines up to replace the removed line, and by the ability to erase to the end of an individual line on the screen. Different kinds of character highlighting make it easier to visually separate text and screen formats and help instructions. It is important to know what happens to the cursor when it attempts to move off screen at any of the four edges. Assumptions about these edges and the cursor movement from the edges are the most frequent source of problems in programs intended to be configured by the user for his terminal.

Table I lists terminal functions which are most frequently used by software. The table has three columns for each feature listed. The first column just indicates whether or not the function is used in some way by the software. The second column allows entry of the number of characters allowed for in the installation procedure for the software. These characters constitute the string which will be sent to the terminal by the software when the function must be performed. The third column provides a place to note whether a means is documented to introduce a patch routine which will handle special problems at a difficult installation.

The list of functions is separated into three groups. The first group consists of the screen height and width parameters along with the direct cursor positioning command. These two parameters are considered essential because they alone are sufficient to give software complete control of the terminal. MicroPro's WordStar (TM) is an example of the use of these features alone to control the terminal. Other terminal functions can be utilized to provide better performance if they are available. But if an installation's terminal has direct cursor positioning, WordStar can be run on it. The second group includes those cursor control features which might be used by software in addition to, or instead of direct positioning. In my experience, this group of controls is associated with assumptions frequently made by software authors which cause the most problems in using their software at many sites. The last group are usually offered as options which, if implemented, enhance performance of the software.



(continued next page)

Allows patching special subroutine			
Number of characters allowed for control string			
Feature used by the software			
Terminal features			
Essential features			
screen height and width definable			
Cursor positioning:			
allows ASCII digits for line&col			
allows separate leadin for l&c			
allows either line/column first			
Alternate cursor positioning			
cursor up,down,left,right			
clear screen			
home			
backspace			
carriage return			
linefeed			
Other features which might be used to enhance performance			
Erase to end of line			
Delete cursor line & move lines up			
Insert blank line at cursor			
Highlighting on/off			
Terminal initialization			
Session termination			
Transmission delays			
Display at last column of last line			
Delete/Backspace defeat			

The cursor positioning command is usually a string of from 3 to 8 characters with the row and column position appearing at various places in the string. The row and column position characters are frequently composed of a position number added to an offset to put the number into a printable character range. Not all terminals use a continuous range of such codes. The ANSI standard for cursor positioning requires that the line and column positions be ASCII numeric characters. A few terminals expect the line and column to be sent with separate lead in strings. Terminals also vary in expecting the line or column to be sent first in the string.

Because the behaviour of the cursor as it leaves any of the four edges of the screen varies considerably from terminal to terminal it is necessary to discuss the cursor positioning in relation to these edges. Assumptions about this behaviour are most distressing to someone trying to install software for a particular terminal. Typically, if the cursor goes beyond the last column of a line it ends up on the first position of the next line. However, other behaviour is possible and not uncommon. The most notable alternative is that the cursor is not allowed to leave the line without special positioning commands. All overflow characters are printed on top of each other at the last character position. Some terminals do this line wrap-around going to the right but not to the left. Behaviour at the top and bottom of the screen also varies. Frequently printing a character at the lower right hand corner position causes the screen to scroll up and the cursor is positioned on the left of a blank line at the bottom of the screen. Again some terminals do not scroll while others offer scrolling and none scrolling modes. The up, down, left, right, backspace and linefeed functions from the group are directly involved with these behaviour differences. The direct cursor positioning function can also be affected by edge

behaviour when the string to be displayed or the input which is expected exceeds the line length. WordStar avoids problems by tracking each character printed or input and by making what-to-do-next decisions when the end of line is reached.

The clear screen and home functions are related because a clear screen function usually does a home also. The Digital Equipment VT52 terminal is an exception because you must independently do a home and then do a clear to end of screen. The Zenith H89 has functions similar to the VT52 but also provides a combined home and clear screen. There are a few terminals which do not recognize the ASCII backspace character 08 to move the cursor left. Using this character within a program and not allowing it to be configured at installation time is a source of problems.

Software is used to send out null characters after the carriage return sequence on printing terminals, to allow time for the mechanical return of the carriage. CRT terminals do not require this at low speeds, but most terminals do require delays for certain commands (such as clear screen) at transmission speeds in excess of 9600 baud. **(In many cases terminals running at 19200 baud will require delays calling for 30 to 50 nulls. Some commands may require delays in mid-sequence; in this case nulls would corrupt the command string. Furthermore, 50 nulls will provide a seemingly interminable delay if the program is ever accessed at 300 baud via modem. A more general and elegant solution to the problem of flexibly providing necessary timing delays is to allow software delay loops to be selectively placed either before, during, or after an escape sequence. Editor.)** I have observed a lot of configuration tables which do not allow enough characters for the command sequences on some terminals. The Digital Equipment VT-52 terminal requires four characters to clear the screen. The cursor is sent home with a two-character sequence and then a clear to end of screen sequence of two-characters is required to complete the job. A few terminals require as many as eight-characters for some of the command strings.

Some terminals provide an automatic linefeed character upon receiving a carriage return. This is usually an option which can be switched off at the terminal. Thus software usually sends a carriage return then a line feed. Not all terminals behave well even for this basic operation, however. The SOL20, for instance, likes to have the line feed sent first. Sending the carriage return first erases the line the cursor is leaving.

There are a few new terminal models which are too intelligent for many applications. They have so many modes and special functions that software is always tripping over them. These terminals require special initialization and sometimes de-initialization to achieve predictable behaviour.

In order to provide backspacing instead of echoing deleted characters some special versions of CP/M provide a backspace-space-backspace in response to the character sent to the terminal following the entered backspace or delete. This interferes with cursor positioning as a response from software. WordStar provides a defeat for this feature on systems which require it.

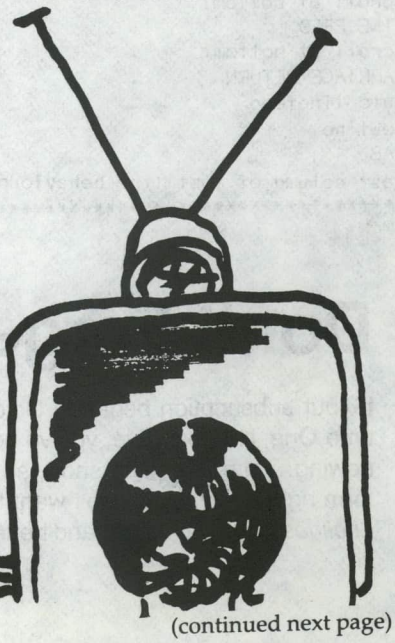
Discussion of the terminal features table:

The following terminal features table puts all these functional details in one place for software installation and terminal comparisons. If several models of one manufacturer have similar features or if some models have a subset of features of another, they can be listed together. Likewise, models of any manufacturer which completely emulate the model being discussed should be listed as a related model.

The line length describes the number of characters which are displayed on a line. The number of lines are the displayed lines. The character set will probably be 64 ASCII characters or 96 ASCII or 96+ ASCII. The "+" indicates special graphic characters or displayable control characters. Line wrap around indicates that if the cursor is advanced past the end of a line it goes to the beginning of the next line. Scrolling describes the action of moving the displayed lines up to make a new blank line at the bottom of the screen when the cursor attempts to move down off the bottom edge of the screen. Some terminals also scroll down at the top of the screen. There are several types of display accenting in use. Color change can be listed under reverse video or graphics. A few terminals perform line insertion and deletion. This usually is accompanied by scrolling up or down. Erasing all or part of a line is sometimes available. The bell is usually either a speaker or a buzzer. Details about the code to use to invoke the various cursor control features and remarks about how they behave are provided for by the last section of the form.

In order to assist the discussion of the installation of software for different terminals, a completed table will be included for each terminal considered in each article of this series along with a completed Table I for each software package discussed.

(See next page for the terminal features table.)



(continued next page)

```
*************************************************************
*      TERMINAL FEATURES                                    *
*************************************************************
* Terminal Model & Manufacturer:                           *
*                                                          *
* related models:                                          *
*                                                          *
* Screen Format-                                           *
*       Line length:                                       *
*       Number of lines:                                   *
*       Character set:                                     *
*       Line wrap-around                                   *
*       Scrolling:                                         *
*                                                          *
* Display Accenting-                                       *
*       Underline:                                         *
*       Reverse video:                                     *
*       Intensified video:                                 *
*       Graphics:                                          *
*                                                          *
* Editing-                                                 *
*       Insert line:                                       *
*       Delete line:                                       *
*       Erase line:                                        *
*                                                          *
* Bell or Buzzer-                                          *
*                                                          *
* Features which effect cursor position-                   *
*----------------------------------------------------------*
* feature                          code        remarks     *
*----------------------------------------------------------*
* HOME                        .          .                 *
*CLEAR SCREEN                 .          .                 *
*clear to end of screen       .          .                 *
*GOTOXY                       .          .                 *
*    uses ANSI Standard       .          .                 *
*    line or column first     .          .                 *
*    command form             .          .                 *
*    offset for binary numbers .         .                 *
* erase to end of line        .          .                 *
* CURSOR LEFT                 .          .                 *
* backspace                   .          .                 *
* CURSOR RIGHT                .          .                 *
* CURSOR UP                   .          .                 *
* scroll at top?              .          .                 *
* CURSOR DOWN                 .          .                 *
* scroll at bottom?           .          .                 *
* LINE FEED                   .          .                 *
* scroll at bottom?           .          .                 *
* CARRIAGE RETURN             .          .                 *
* auto linefeed               .          .                 *
* newline                     .          .                 *
* TAB                         .          .                 *
* last column of last line behaviour                       *
*************************************************************
```

Don't Forget...

If your subscription began in October of 1980 with Volume One, Number Five, you've heard from us about renewing. Time is getting short, so send in your renewal form right away. You won't want to miss a single issue; *Lifelines* is getting bigger and better all the time.

Calling SORT As A Subroutine From MBASIC SORT

This information comes to us courtesy of MicroPro International Corporation.

Editor's Note: While this application note is specific to the SUPER SORT/BASCOM combination, the same approach can be generally applied to assembly language interfaces to compiled languages. Lifelines plans in the future to print an article fully describing the dos and don'ts of interfacing to and between the wide variety of languages available for micros.

This application note describes a means whereby SUPER SORT may be linked with MBASIC rel files produced by the Microsoft BASIC Compiler. It assumes the use of SUPER SORT I's SORLIB and Microsoft BASIC development package including the BASCOM BASIC Compiler, the M80 macro assembler and the L80 linking loader.

Because of MBASIC's limited programmer control over the organization of variable length and variable type data items, this note will concern itself with an assembly language interface to the SORSUB sorting routine in SORLIB. This assembly language module, which is linked with and called by the MBASIC program, provides all the parameters required by the SORSUB sorting module to effect a single sort operation. All the sort/merge file names, input attributes, output attributes, key/select/exclude options, and user exit routines are specified in this module after which control is transferred to the SORSUB routine. After sorting/merging is complete control is passed back to the MBASIC main program. All the sorting options to be invoked in the assembly language module are described in the SUPER SORT Programmer's Guide. Multiple modules may be written to support user interactive sorting; it must be noted, however, that all sort specifications are frozen at link time. This means that all of the parameters such as record length, sort keys and file names cannot be changed during the execution of the program.

Passing Control to SUPER SORT

In this example a simple call without parameters executed from the BASIC program

passes control to the assembly language module which in turn calls the SORSUB sorting subroutine. The BASIC program looks like this:

```
100 CALL SORT1
```

where SORT1 is declared as an entry point in the assembly language routine that is setting up the parameters for SORSUB.

See Listing I for the assembly language module. Listing II is an example for defining SUPER SORT's work space in the assembly language module.

Parameter Block Definition

The parameter block for SUPER SORT contains all the information needed to perform a sorting operation. When SORT.COM is executed an equivalent of the parameter block is created as the operator inputs to the keyboard, or as a CFILE is read. When the REL file version is used, however, the programmer must construct one in memory. When using the MBASIC Compiler it is advised that the parameter block be specified in assembly language. Although it may be argued that an MBASIC array may be used to construct the parameter block the mechanism to do so is more complicated than to write it in assembly language. Also if file size is a consideration assembly is much to be preferred. To facilitate the construction of the parameter block a sample listing is provided. (See Listing III.) The programmers' guide section of the SUPER SORT manual also contains a detailed description.

Listing III shows a parameter block example which does a simple sort of the input file CUSTOMER.DTA and produces the output file NAMES.DTA.

Multiple Assembly Language Interfaces

The above procedure should be sufficient for simple applications. In many cases, however, it is necessary to initiate more than one type of sort from the BASIC program. This can be most simply handled by coding two separate assembly language interfaces and then calling the appropriate one from BASIC. Each module, once it had set up the parameters, would then call the SORSUB sorting subroutine. An illustration of the BASIC program segment follows:

```
100 IF A%=1 THEN CALL SORT1
200 IF A%=2 THEN CALL SORT2
```

where SORT1 and SORT2 are the entry points to two separate assembly language modules each of which has its own parameter block. When using this method space can be saved by defining the work space in only one of the modules. This is done by declaring the label at the beginning of the work space to be a public label to other modules. The label is then referenced by the other modules as an external label. It is important to never specify a work length which is longer than the actual length of the work space in the defining module.

Parameter Passing from MBASIC

In some applications which require more than one type of sort operation the differences between the two sorts is so trivial that there is no justification for constructing two separate parameter blocks. Let us consider an application which involves two different key fields. In this instance it is only necessary to pass the changing parameter in the CALL to the assembly language module. The assembly language module then does the actual modification to the parameter block and the call to SORSUB is made as described above. An example in BASIC and the beginning of the assembly language segment to handle the parameter pass is shown below:

```
50 REM I% REPRESENTS THE FIELD TO BE USED AS THE KEY
 .
 .
100 I%=1 :REM USE FIELD 1 AS A KEY
110 CALL SORT(I%)
 .
 .
200 I%=5 :REM USE FIELD 5 AS A KEY
210 CALL SORT(I%)
 .
 .
```

The assembly language segment looks like this:

```
SORT  EQU    $
      ;      HL REG. CONTAINS A POINTER TO THE
      ;      PARAMETER WHICH REPRESENTS THE KEY
      ;      FIELD OF THE SORT
      ;
      MOV   E,M    ;get low order byte of parameter
      INX   H
      MOV   D,M    ;get high order byte of parameter
      XCHG         ;actual parameter is in HL
      SHLD  KEYFLD ;put it in the parameter box
      ;
      ;...and so on....
```

where KEYFLD is the label of the point in the parameter block where the key field is located. For a complete description of parameter passing in MBASIC refer to the Assembly Language Subroutines appendix to the MBASIC manual.

Linking It All Together

Once all of the program segments have been edited, compiled and assembled it is time to link them all together with the L80 linking loader, to create a <filename>.COM file ready for direct execution under CP/M. An example of linking a BASIC program with an assembly language interface which uses calls to the SORLIB SUPER SORT library goes something like this:

```
A>L80

(L80 sign on message)
```

(continued next page)

```
*CUSTSORT,BASLIB/S,SORT,SORLIB/S,CUSTSORT/N/E
```

Where:

CUSTSORT is the user supplied BASIC program

BASLIB/S initiates a search through the BASIC library supplied with the compiler to satisfy undefined labels.

SORT is the assembly language module which contains a call to global labels in the SORLIB library.

SORLIB/S initiates a search through the SORLIB library to satisfy the remaining undefined labels.

CUSTSORT/N/E directs the linker to wrap things up and save the result in CUSTSORT.COM.

If there is more than one BASIC program being linked they should all go before BASLIB/S on the command line. In the same way all assembly language modules, if more than one, must go before SORLIB/S.

Conclusion

The preceding application note is by no means a complete description of the possibilities of MBASIC calling sequences to SORSUB. The basic technique of using an assembly language interface to effect the call, however, should prove flexible enough to accommodate most all applications.

LISTING I

The assembly language module looks like this:

```
        ENTRY   SORT1           ;declare the entry and external labels
        EXT     SORSUB,$MEMRY   ;*** SEE NOTE BELOW FOR $MEMRY ***
        ;
SORT1   EQU     $               ;this is where control passes from BASIC
        ;
        LHLD    $MEMRY          ;get the location of the beginning of free space area
        XCHG
        LXI     H,PARBLK        ;location of the parameter block
        LXI     B,SORARS        ;location of pointers  -work length
                                ;                      -status return
                                ;                      -altseq table
        CALL    SORSUB          ;call to the SUPER SORT routine
        RET                     ;return to BASIC
SORARS:         ;pointer area
        DW      WLEN            ;pointer to the work area length
        DW      SSTAT           ;pointer to the status return variable
        DW      COLTAB          ;pointer to alternate sequence table
WLEN:   DW      0               ;actual work length area (0 indicates all free space)
SSTAT:  DS      2               ;make room for status return
COLTAB  EQU     $               ;alternate collating sequence would go here if needed
PARBLK  EQU     $
        ;
```

$MEMRY NOTE: In order for SUPER SORT to work at maximum speed a large work space is desirable. In this example the symbol $MEMRY (generated by L80 during linkage) contains a pointer to the top of the program being linked. Unfortunately MBASIC uses the free space area for storage of string variables and possibly for other purposes. This means that if your program uses strings or you experience inexplicable results after a sort which uses the free space area for work space then you must set aside another place for SUPER SORT to work.

LISTING II

Below is an example for defining SUPER SORT's work space in the assembly language module:

```
        ENTRY   SORT1           ;declare the entry and external labels
        EXT     SORSUB
SORT1   EQU     $               ;this is where control passes from BASIC
        LXI     H,PARBLK        ;location of the parameter block
        LXI     D,WORK          ;location of the beginning of the work space
        LXI     B,SORARS        ;location of pointers  -work length
                                ;                      -status return
                                ;                      -altseq table
```

```
        CALL    SORSUB          ;call to the SUPER SORT routine
        RET                     ;return to BASIC
        ;
        ;
SORARS:         ;pointer area
        DW      WLEN            ;pointer to the work area length
        DW      SSTAT           ;pointer to the status return variable
        DW      COLTAB          ;pointer to alternate sequence table
WLEN:   DW      1000H           ;actual work length area
SSTAT:  DS      2               ;make room for status return
COLTAB  EQU     $               ;alternate collating sequence would go here if needed
        ;
WORK:   DS      1000H           ;make room for the work space
        ;
PARBLK  EQU     $               ;parameter block starts here
```

LISTING III

This parameter block example does a simple sort of the input file CUSTOMER.DTA and produces the output file NAMES.DTA:

```
;       ****************************************
;       *  BEGINNING OF SORSUB PARAMETER BLOCK *
;       ****************************************
;
;
PARBLK  EQU     $
IRECL:  DW      128             ;maximum record length for input files
CRECL:  DW      0               ;record length for output file
                                ;           0=same length as the input file
        DB      0,0,0,0         ;reserved
ICRDF:  DB      1               ;input file type
                                ;  1=cr-delimited
                                ;  2=fixed-length
                                ; 64=variable
                                ;128=relative
IEOF:   DB      0               ;input files end of file option
                                ;0=default
                                ;1=no-single-z (sum 1 and 2 for variable and relative type
                                ;2=no-zzz
                                ;4=ffzzz
CFNAME: DB      0,'NAMES  DTA   ;output file name, (FCB format)
OFOP:   DW      0               ;for output disk chage bit 15=on (80H)
OCRDF:  DB      1               ;output file type (same options as ICRDF)
OEOF:   DB      0               ;output file end-of-file option
                                ;0=normal
                                ;2=ffzzz fill
WDRV:   DB      0               ;work file driv
                                ;0       =current logged drive
                                ;1 or 'A'=A:
                                ;2 or 'B'=B: etc.
RNOUTF: DB      0               ;non-0 for R-OUTPUT
KANOUT: DB      0               ;non-0 for KR-OUTPUT
TAGSF:  DB      0               ;non-0 for TAGSORT
        DB      0               ;reserved
KONLYF: DB      0               ;non-0 for K-OUTPUT
KPOUTF: DB      0               ;non-0 for KP-OUTPUT
POUTF:  DB      0               ;non-0 for P-OUTPUT
XIT1F:  DB      0               ;non-0 to invoke use of user installed XIT1
XIT2F:  DB      0               ;non-0 to invoke use of user installed XIT2
PRNLVL: DB      5               ;print level 0 (nothing printed) to 5 (full print)
        DB      0               ;reserved
;
;       THIS MARKS THE END OF THE FIXED LENGTH SECTION OF THE PARAMETER BLOCK
;
```

```
;       ********************
;       * SORT INPUT FILES *
;       ********************
        ;
        ;
NOSORFL: DW     1               ;number of sort files, 1-32.
                                ;may be 0 if merge-only files are given
        ;       THE NEXT FOUR ITEMS ARE REPEATED FOR EACH SORT INPUT FILE
        ;                       OMIT IF NONE
        ;
        ;*****************************************************************************************
        DB      0,'CUSTOMERDTA' ;sort input file drive and name (FCB format)
        DW      0               ;reserved
        DW      0               ;starting record (0=beginning of file)
        DW      0               ;ending record (0 or OFFFFH=entire file)
        ;*****************************************************************************************
        ;
        DW      OFFFFH          ;mark end of input sort files
        ;
;       *********************
;       * MERGE INPUT FILES *
;       *********************
;
;
NMOFL:  DW      0               ;number of merge-only input files, 1-32.
                                ;may be 0 if sort-only input files are given
        ;
        ;
        ;       THE NEXT TWO ITEMS ARE REPEATED FOR EACH MERGE INPUT FILE
        ;                       OMIT IF NONE
        ;            (delete ';' if merge files are used)
        ;
        ;*****************************************************************************************
        ;DB     0,'MERGE  DAT'  ;merge-only file name, (FCB format
        ;DB     0,0,0           ;reserved
        ;DB     0,0,0
        ;*****************************************************************************************
        ;
        DW      OFFFFH          ;mark end of merge input files
;       **************
;       * KEY FIELDS *
;       **************
;
NKEX:   DW      1               ;number of key field: 1 to 32
        ;
        ;       THE NEXT THREE ITEMS ARE REPEATED FOR EACH KEY
        ;                       MUST BE AT LEAST ONE KEY
        ;*****************************************************************************************
        DW      1               ;start column (positional field)
                                ;or field number (comma-delimited field)
        DW      30              ;field length in bytes (maximum length for comma-delimited field)
        DW      1               ;field type and attributes (additive)
                                ;
                ;ATTRIBUTE              DECIMAL         HEX             BIT #
                ;comma-delimited        1               1               0
                ;numeric-ascii          2               2               1
                ;upper-case             8192            2000            13
                ;right-justify          64              40              6
                ;lo-hi                  4               4               2
                ;mask-parity-bit        16384           4000            14
                ;ebcdic                 20480           5000            12+14
                ;altseq                 1024            400             10
                ;twos-complement        32              20              5
                ;integer                36              24              2+5
                ;floating-point         16              10              4
                ;packed bcd             128             80              7
```

```
                ;descending
                ;(omit for ascending)        256             100         8
;*****************************************************************************************
;
                DW      0FFFFFH         ;mark end of keys
                ;
                *****************
;               * SELECT FIELDS *
;               *****************
;
NSEL:           DW      0               ;any value 1-32 to use record selection
                ;
                ;       PROPERLY FORMATTED SELECT SPECIFICATION STRINGS WOULD BE INSERTED HERE. THE FORMAT
                ;       FOR RECORD SELECTION IS FULLY DESCRIBED IN SORSUB RECORD SELECTION SECTION OF THE
                ;       SUPER SORT PROGRAMMING GUIDE.
                ;
                DB      0FFH            ;marks end of record selection string
                ;
;               ***********************************************
;               * THIS IS THE END OF THE SORSUB PARAMETER BLOCK *
;               ***********************************************
                ;
                END
```

A Reader's Comment

To the Editor:

I really have nothing but admiration for James R. Reinders [sic] industry in programming a true random function for Microsoft BASIC. I do find fault, however, with the concept of taking a 747 jet liner to get from Brooklyn to the Bronx.

I am, therefore, enclosing my BASIC Sub-routine for arriving at a true random number, in the interpreted or compiled version of BASIC-80, which is shorter, simpler and much more versatile.

```
10   'Random Function Test
20  '
25  DEFINT
30  GOSUB 180
40  PRINT INT(RND*100)
50  END
60    '    *********************************
180 PRINT"    PRESS THE SPACE BAR WHEN YOU ARE READY    " :PRINT
181 N=RND
182 N=N*2 : IF N>32000 THEN N=N/3
183 IF INKEY$<>CHR$(32) THEN 182
184 N=INT(N) : RANDOMIZE(N)
185             RETURN
```

Bob Kowitt
The Computer Consultant
E. Meadow, N.Y.

Reply

Mr. Kowitt's letter does indeed represent a simpler way of getting to the Bronx - (far superior to the subway which I utilize daily). It has the advantage of being comprehensible ..(the subroutine, not the subway system).. by almost any Basic programmer, and has the potential to return a more 'random' seed. Unfortunately, it does not do what Mr. Reinders most wanted his subroutine to do - provide a random number entirely by the computer - untouched by human hands. Mr. Kowitt's solution is more elegant than Microsoft's, but it suffers if a long running program calls the subroutine many times but otherwise has no need for console input. This might then force some unfortunate person to stare at a tube for hours just to strike a key on cue. (If the computer released a pretzel each time...)

Note: Mr. Reinders's method is adequate for most purposes, but it will not return the best possible pseudo-random seed. This may be demonstrated by running a program which does the following:

Modify the subroutine so that only the contents of the R register is returned. Call the subroutine N*128 times, (where N may be any positive number, and 2 < N < 50 works nicely). Count the number of occurrences of each different returned value. (The program could say something like LET NUM(RV)=NUM(RV)+1, where RV is the returned contents of the R register). If the array is then printed, very noticeable patterns will usually be seen.

b.r.n.

8080 Programming Tutorial: Introduction and Terminology—Part 1

by Ward Christensen

PART I
INTRODUCTION

Why a Tutorial?

There are still people just getting into microcomputing. Perhaps they have been in it for a while, but mostly working with BASIC.

The programs on the CPMUG disks draw a lot of attention, and frequently people would like to modify, or at least read, some of the assembler programs on those disks.

Why 8080?

As a hobbyist's and general purpose microprocessor, the 8080 has now been around for about 6 years. The compatible, but more "complete" Z-80 microprocessor might be considered by some as a more likely processor to teach.

However, the 8080 is still the "lowest common denominator", and the most popular processor for assembler programs in The CP/M Users Group.

Along with the fact that I am not thoroughly expert in the Z-80, the availability of a standard assembler, ASM, supplied with CP/M, justifies the 8080 basis for this tutorial. I strongly encourage someone else to make use of their Z-80 knowledge, and write a tutorial using the Zilog mnemonics.

Background

I have written hundreds of 8080 assembler programs. Most are of a simple utilitarian nature to solve a problem I might have. Others have become more widespread in their usage, such as MODEM for transferring files over phone lines, DU, the disk utility, and FMAP, UCAT, and CAT, the master cataloging system programs.

As some of you know from reading my previous Lifelines articles, my style is rather informal. I will teach, not from books, but from experience. My terminology and definitions come from everyday usage, not "dictionary definitions". I will attempt to give enough information for the novice to pick up, at least, program reading, if not program writing.

I encourage comments: send them to me in care of Lifelines.

This tutorial was originally printed in the CACHE Register, the newsletter of CACHE, the Chicago Area Computer Hobbyists Exchange.

In addition to a general going-over (adding new things I have learned), I will add sections to it specifically related to CP/M, which were not covered in my original articles.

HIGHLIGHTS

The tutorial will:

* present necessary terminology (expanded upon greatly from the CACHE version)
* cover all the 8080 instructions
* introduce programming tips
* warn of the easy "pitfalls" which plague the beginning 8080 programmer
* "explore" a typical CP/M program which makes use of DISK Input/Output

TUTORIAL OUTLINE

The following topics will be covered in the 8080 tutorial:

1. INTRODUCTION
2. TERMS - an introduction to general-use microcomputer terms, and the terms which will be used in the following sections.
3. 8080 ARCHITECTURE - as need be known to program it.
4. THE ASSEMBLER - How to prepare assembler programs for processing by the CP/M assembler. Use of comments, etc.
5. DATA MOVEMENT INSTRUCTIONS - the simplest instruction for moving data within the 8080 and between the 8080 and memory.
6. ARITHMETIC INSTRUCTIONS - doing add, subtract, increment and decrement.
7. THE 8080 STACK - Subroutines, saving data, PUSH and POP instructions.
8. CONTROL OF EXECUTION SEQUENCE - the JMP, CALL and RET instructions, and their conditional forms.
9. INPUT OUTPUT INSTRUCTIONS - getting data into and out of the computer.
10. OTHER INSTRUCTIONS - The 8080 instructions which do not fall into the above groupings.
11. PITFALLS: while not a very large section, it will cover the more common mistakes beginning 8080 programmers make, (Mostly because I remember making them!)
12. SUBROUTINES - Some useful routines which accomplish what a single instruction cannot: message printing, base conversion (decimal-binary) etc.
13. CP/M INTERFACE - Tying it all together.

REQUEST FOR QUESTIONS

After the final installment of the tutorial, I would like to continue with a question and answer column, if there is sufficient interest. As soon as something catches your interest, jot it down and send it to me C/O Lifelines. Questions of general interest (which I am able to answer) will be reprinted in Lifelines. Personal answers will be limited by a time-available basis, and will be to those questions accompanied by a stamped return envelope (or post card).

Does anyone have a suggestion for which program to "dissect"? It should exercise many of the CP/M capabilities, yet should not be too long. I'd prefer it to be one of my own CPMUG contributions, as that would mean I can best explain not only the instruction functions, but why I chose to do something a particular way.

People often ask me to recommend books which they may use to learn 8080 programming, to be able to read and/or modify the programs in the CPMUG. Does anyone know of such a book that I might recommend? I'll pass it on via this column, too.

PART 2 - TERMS

Knowledge of the following terms will help you in getting through the future sections of this 8080 tutorial. I frequently find it necessary to use new terms in the definitions of others.

Therefore, when using one of the defined terms for the first time in a pararaph, I will usually make it UPPER CASE.

The terms are defined as they will be used in the tutorial, thus specific references to the 8080, and CP/M will be found in many of them. However, most of the terms apply more generally. These are not "official" definitions, but just "common usage" - as have become common to microcomputerists over the years.

I hope through (sorry, you "through" lovers) the tutorial will build a general "microcomputerists vocabulary", since programming frequently involves discussions such as "will you be running at 4MHz?" and "does your printer hook to a serial, or parallel port?", so it helps to know these terms also.

At times, the explanations get very "basic", i.e. you may say "I know that". However, there ARE people who read 8080 programs, who have asked me things like

He: "I don't understand some of the commands"
Me: "What do you mean?
He: "Ah, like MVI.
Me: "Oh, you mean 8080 Op Codes
He: "Oh, is THAT what you call them?

Thanks to Chuck Weingart, Russell Stevens, and Joe McGuckin, for suggesting many of the terms which I didn't think of.

ACCUMULATOR

A special REGISTER in which arithmetic and logical operations are performed.

In the early days of computing, the electronic circuits required to make an accumulator were very expensive, so computers typically only had one. Although that reason is gone, many computers still have only one, or a limited number, of accumulators. Another reason for the limitation is that their address (i.e. REGISTER) is implicit and not explicitly stated in the instruction.

For example, in the 8080, the instruction ADD B implies that the sum will be in A, the accumulator. In the 8080, the accumulator is 8 BITS, or 1 BYTE wide.

ADDRESS

The MEMORY of a computer may be considered to be divided into locations, each of which has an ADDRESS. The first ADDRESS in the computer is 0, the highest depends upon how much MEMORY is in the computer, but has a maximum of 65,536 in the 8080. Each of these MEMORY locations is called a BYTE (see BYTE).

ADDRESS also applies to the location which an external device talks to the 8080 - via a PORT.

AND

Used in programming, much as it is in common English: to mean "both" of two conditions must be true. For example, "I will go to the club meeting if I can get a ride, AND if I don't have to work that day".

AND also refers to the combining of BITS, in a "both" fashion:

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

In an 8080, we are always ANDing 8 BITS at a time. For example, to test if a particular BIT is on in a BYTE, we AND that BYTE with another BYTE. The results will be zero only if the tested bit was zero. Otherwise, it will have the value of the tested bit, although usually we are just concerned whether it is zero or non-zero.

For example, to test if bit 2 is on (see BIT for bit numbering), we AND with a byte which has bit 2 on also.

Thus, to test the byte:

X X X X X 1 X X

to see if bit 2 is on, we AND it with the value:

0 0 0 0 0 1 0 0

producing the result:

0 0 0 0 0 1 0 0

Since this value is non-zero, we know the 2-bit of the original byte was on. If it had been OFF, the result would have been:

(continued next page)

0 0 0 0 0 0 0 0

Conceivably you could test multiple bits at once, getting a non-zero result if any of the bits were on, but I have not found a need to do this.

ASCII

An acronym for American Standard Code for Information Interchange. This is a standard code for representing letters, numbers, punctuation, special characters, and control characters as a 7 BIT value. In a MICROCOMPUTER the characters in the ASCII code occupy the right 7 BITs of an 8 BIT BYTE.

ASSEMBLER

A PROGRAM which converts a SOURCE PROGRAM consisting of INSTRUCTIONS (OP CODE and OPERAND) into an OBJECT PROGRAM in 8080 internal format. For example, to move a value of an ASCII '8' into REGISTER A, you code:

```
MVI A,'8'
```

in your program. The ASSEMBLER then converts this to "3E 38" (HEX) which is the form the 8080 understands it in.

The word ASSEMBLER also refers to the language of writing PROGRAMs, as used in the phrase "I wrote the program in ASSEMBLER" (as opposed to BASIC or FORTRAN, two other computer languages).

The reason for the interest in assembler programming is that assembler programs generally run faster, and take less room, than programs written in higher level languages.

Also there are certain functions such as INTERRUPT handling which are either impractical or impossible in higher level languages.

The disadvantage is the knowledge and time required, and the increased difficulty (read "easier to screw up") of assembler programs. See the following item also, as "ASSEMBLY LANGUAGE", and "ASSEMBLER" are often used interchangeably.

ASSEMBLY LANGUAGE

Sometimes simply called "assembler": The format and content of the programs which are processed by the ASSEMBLER.

This includes LABELS, OP CODES, OPERANDS, and COMMENTS.

ASYNCHRONOUS

Usually refers to events, i.e. two events which do not need to occur at the same time or at some fixed repeating interval, are ASYNCHRONOUS.

The term occurs most commonly in reference to SERIAL data transfers, i.e. in which the BITS of a BYTE are sent one-at-a-time.

ASYNCHRONOUS (often abbreviated ASYNCH) data transfer means the time at which each BYTE of data starts, is independent of when the previous character had started. Another way of saying that, is that there is no predetermined time between characters.

BAUD

Refers to the number of BITS per second, at which a SERIAL TERMINAL operates. Without consequence, you may just take BAUD it to mean the same as "Bits per second". I.E. a 300 BAUD terminal transfers data at 300 bits per second.

I don't want to get into a long discussion of this, but there is a minor distinction between "bits per second" and "baud", but it only comes into play for example, when you deal with some "bits" which take 1.5 time units.

BINARY

The number system in base 2. The number of digits used in any base is 1 less than the base. Base 10 uses 10 digits, from 0 through 9. Base 2 uses 2 digits, 0 and 1.

Base 2 is frequently used in computers because of the capability of an electronic device (usually a transistor) to be either conducting, or not conducting electricity.

Also common to different bases: In a multi-digit number (such as 297) the rightmost digit stands for units ('ones'), the next digit to the left stands for the base, and the next for the base squared, etc.

This is true for base 2, as it is for any other. The rightmost digit stands for units (0 or 1), the next, twos (again 0 or 1) the next 2 squared, or 4, etc.

If we think of a BYTE as being an 8 BIT number, then we count the bits from left to right having the following values: 128, 64, 32, 16, 8, 4, 2, 1. Thus, the number 128 is represented in base 2 as 1000000, the number 5 as 00000101. The '101' means 1 four, plus 1 one, or 5.

Because it is difficult to deal with binary numbers directly, they are frequently grouped in threes or fours, which results in either an OCTAL (when grouped in threes) or HEXADECIMAL (usually just called HEX, when grouped in fours). Thus 5 in OCTAL would be grouped:

```
00 000 101
```

and in HEX would be grouped

```
0000 0101
```

We then give the values for the groupings, instead of the BITS. Thus in OCTAL we have 005, and in HEX we have 05. The decimal number 100 is represented in binary as 01100100, which in OCTAL (grouped 01 100 100) is 144, and in HEX (grouped 0110 0100) is 64.

(Next month this multi-part tutorial will continue with more terminology.)

Assembly Language Interface to PL/I-80

by Michael J. Karas

The advent of the PL/I-80 high level programming language for CP/M-based machines has opened up the wide wide world of easy, sophisticated, and structured programming for business, commercial and scientific applications. The language features offer the most capable development tools of any microprocessor programming language available in the marketplace today. Through extended use of the language, others, like myself, are going to find out that the features of PL/I-80 offer far more than just programming nicety for applications software development. The structure, speed of execution, self-documentation features, and full CP/M compatibility make PL/I-80 the obvious choice for many systems utility and dedicated function processor programming exercises.

The one main disadvantage of any high level language is the isolation from the "guts" of the computer machinery that results for the programmer. Don't get me wrong by thinking that this is altogether a disadvantage. After all, it is the whole idea behind the use of a high level language in the first place. But for certain aspects of systems utility and dedicated function processor programming, more access to the internals of the computer are generally needed. The normal manner of obtaining this capability is through development of programs in machine language. But we all know what a pain that can be for very large programs thousands of bytes in size. It always seems that when an assembly language program gets over about 2000 bytes, that there is a screaming need for easy access to disk files, operator interfaces, and simplified development of complex logical structures. PL/I-80 offers all of the things the heavy duty assembly language programmer needs to make life easy. The next obvious question is... How do I handle that special interface?..or How can I do IN and OUT in PL/I-80?..or How can PL/I-80 access the SYSTEM tracks of a CP/M floppy disk?...and the list of questions goes on and on and on. The answer to all these questions is to use the assembly language interface facilities offered through the mechanism of external procedure calls in PL/I-80.

The Digital Research LINK-80 manual describes how it all works but doesn't make it very easy to understand. Here in example form I intend to present an easy explanation of how to implement an assembly language interface to PL/I-80. I have chosen to implement "hooks" to allow direct access to the CP/M BIOS vector table of CP/M 2.2 so that the PL/I-80 programmer can make those special system utility packages with a minimum of pain, yet leave the avenue open to perform all those special I/O functions that the standard CP/M BDOS interface never seems to have enough of. I have chosen to write a diskette copy utility program in PL/I-80 and to let the compiler make it easy to write all the operator prompting messages and to allow quick human readable presentation of the logical structure of the program, while at the same time allowing direct high speed access to the disk drivers in the BIOS.

The start of this project resulted in the assembly language module given below. This module provides interface translation between the external procedure calling mechanisms of PL/I-80 and the entry/exit requirements of the various assembly language hardware interface drivers of the CP/M 2.2 BIOS. Detailed discussion of all aspects of the interface conventions are well beyond the scope of this article but a few general comments are in order. Readers who desire to "understand to the MAX" are encouraged to A) study the examples given here carefully while B) consulting the PL/I-80 LINK-80 Users Manual and C) consulting the CP/M 2.2 System Alteration Guide. As a start, to become the most familiar with the mechanisms I would suggest implementing the examples given here on your own computer system. It only takes a few days or evenings and the experience can be quite rewarding.

The interface to the BIOS, in general, is done as follows: The location of the BIOS vector table of entry points is determinable in any CP/M system by looking at the address in locations 1 & 2 of low memory. This address gives the pointer to the warm boot address of the BIOS vector table. All other BIOS entry points are accessible by adding an offset to the warm boot pointer vector. Some functions are console status, line printer output, and disk track selection. All of the offsets necessary are given in the assembly language source code below. The typical procedure to get to a BIOS subroutine, with 8080 type machine language, would be like this:

```
function:
  lhld 0001     ;get warm boot vector table pointer
  lxi d,offset  ;get offset for desired BIOS entry pt
  dad d         ;make address of entry point
  pchl          ;branch indirect through (HL) to BIOS
                ;subroutine
```

If the label entry "function" had been CALLed from some main program, then the return at the end of the BIOS subroutine would get program execution back to the calling program, assuming we use the callers stack all along the way. If the entry to the interface routine required post processing of BIOS return parameters before returning back to the main calling program, then the above program segment becomes:

```
function:
  lhld 0001     ;get warm boot vector table pointer
  lxi d,offset  ;get offset for desired BIOSentry pt
  dad d         ;make address of entry point
  lxi d,backhere;put return address to here on stack
  push d
  pchl          ;branch indirect through (HL) to BIOS
                ;subroutine
  backhere:
```

(continued next page)

```
    <.... post processing
              instructions....>
ret             ;back to main program call point
```

All parameter information entered at the BIOS follows the following general set of rules:

a) If entry parameter is a single byte it is placed in the (C) register.

b) If entry parameter is a double byte or an address it is entered in the (BC) register pair.

c) Return parameters of one byte values are brought back in the (A) register.

d) Return address of double byte parameters are returned in the (HL) register.

For now that should give the reader the information needed to understand the BIOS part of the assembly language module given below. The full set of PL/I-80 rules are complicated, so I will only touch on the concept here and the few interface types used by the BIOS module to return parameters to PL/I-80. For character string variables, the return parameter is put on the stack with the length of the string given back in the (A) register. This scheme applies to the CONIN and READER routines which bring a byte character back from the BIOS. These push the character onto the stack and send a 01 length back to the calling PL/I-80 program.

If the return parameter is a PL/I-80 data type that is one byte in size, but is a numerical type variable, then the value is returned to PL/I-80 in the (A) register. If a numerical type variable of two bytes or an address is being passed back then the PL/I-80 program gets the parameter back in the (HL) register pair.

All entry data sent from the PL/I-80 program to the assembly language module comes as follows if a parameter is defined for passage to the BIOS: the rule is that when called, the assembly language program receives an address in the (HL) register pair that points to a table of memory locations (in our case a single pair). These two bytes of memory contain another address that points right at the data parameter. The type, size, and number of parameters are defined by the design of the assembly language program and the statement of external entry point characteristics. No type coding is passed or error check done. In other words, both sides have to agree to what is trying to be done before the call and return parameter passing will work properly.

The program module "PLIBIOS.ASM", given below in source code form, establishes the names for sixteen small programs that translate procedure calls from within a main PL/I-80 program to the forms required by the BIOS. The subsequent module below, "BIOSCALL.DCL" is a small file that fully defines, in a manner compatible with the assembly language program, the form that the PL/I-80 program must use to call the entry point definitions to make the parameter passing work.

The assembly language program is meant to be assembled into a relocatable file of the ".REL" type by use of the Digital Research relocating macro assembler, RMAC. This assembler, seeing the pseudo opcodes "PUBLIC" on the program names builds the .REL file with a table to tell the [later utilized] link program where the relative addresses of the subroutines in the module are located. After the host PL/I-80 program, like the disk copy utility given later in the article, is compiled into a relocatable object module, that .REL module also contains a table of subroutine call addresses which were known to be external to the PL/I-80 program. The external nature of the labels is defined by the included declare statement from the file "BIOSCALL.DCL" which simply told the compiler that I intended to provide another ".REL" module that had these program names in it, specifically "PLIBIOS.REL". The Link program LINK 80 is used to hook the two modules together and make an absolute object module out of them. The absolute object module happens simply to be the executable CP/M type ".COM" file.

File PLIBIOS.ASM

Direct CP/M 2.2 Bios access Interface Module For PL/I-80

This module to be assembled with Digital Research RMAC provides interface conversion between PL/I-80 calls and function references and the Bios vector table entries. The file "BIOSCALL.DCL" should be included in your PL/I-80 source program with a %include statement to properly establish the external entry point names and calling format definition. Most bios entry point routines are treated as function call modules.

The user of these routine in PL/I-80 calls should be aware that some differences exist between the CP/M 2.2 BIOS implementations that are available from various software and hardware vendors. In particular, not all BIOS routines perform sector translation with the SECTRAN routine. Secondly, if the BIOS performs the function of physical sector deblocking, and the sector translate function is not used, then logical sector numbers entered at the BIOS settrk entry point may require numbering from zero instead of the usual start with logical sector number 1. This assembly language routine makes no assumption about the sector number start. Also no translation is done. The calling PL/I-80 program must anticipate the characteristics of the host system BIOS. (BEWARE especially with tricky utilities being upgraded from CP/M 1.4 including Digital Research's SYSGEN program.) Hopefully CP/M 3.0 coming out soon will fully resolve this problem. Note that CP/M 3.0 will probably fully eliminate the need to have direct BIOS access at all.

```
   name    'PLIBIOS'
   title   'Direct CP/M BIOS Vector Calls From PL/I-80'
;
;
;***************************************************
;*                                                 *
;*    bios direct calls from pl/i for direct i/o   *
;*                                                 *
;***************************************************
;
      public  cboot   ;cold boot reload of cp/m entry
      public  wboot   ;warm boot reload of ccp
      public  cstat   ;return byte for console status
      public  conin   ;return character input from console
      public  conout  ;write character to console
      public  list    ;write character to list device
      public  punch   ;write character to punch device
      public  reader  ;return character input from reader
      public  home    ;home selected disk unit
      public  seldsk  ;select disk unit and return
                      ;parameter table pointer
      public  settrk  ;select track
      public  setsec  ;select sector number
      public  setdma  ;set disk i/o data buffer pointer
      public  read    ;read selected sector and return status
      public  write   ;write selected sector and return status
      public  lstat   ;return byte for console status
      public  sectran ;translate sector number
;
;
; Bios vector table offsets from the warm boot vector
; to access the vector table entry points off the
; loc 1 & 2 warm boot vector
;
cbooto  equ     -3      ;cold boot offset
wbooto  equ     0       ;warm boot offset
cstato  equ     3       ;console status offset
conino  equ     6       ;console input offset
conouto equ     9       ;console output offset
listo   equ     12      ;list device output offset
puncho  equ     15      ;punch device offset
readero equ     18      ;reader  device offset
homeo   equ     21      ;disk home offset
seldsko equ     24      ;select disk offset
settrko equ     27      ;select disk offset
setseco equ     30      ;select sector offset
setdmao equ     33      ;set dma address offset
reado   equ     36      ;read logical sector offset
writeo  equ     39      ;write logical sector offset
lstato  equ     42      ;list status offset
strano  equ     45      ;sector translate offset
;
;
; Define position of CP/M 2.2 warm boot vector location
;
wbvect  equ     0       ;addr of warm boot jump
;
;
;***************************************************
;*                                                 *
;*       general purpose routines used upon entry  *
;*                                                 *
;***************************************************
;
;
; get single byte parameter to register c
;
getp1:
        mov     e,m     ;low (addr)
        inx     h
        mov     d,m     ;high(addr)
        xchg            ;hl = .char
        mov     c,m     ;to register c
        ret
;
;
; get single word value to BC
;
getp2:
        call    getp1 ;get low byte of
        inx     h     ;parameter
        mov     b,m   ;get high byte also
        ret
;
;
; transfer ctrl to indexed bios vector table entry
; enter with DE = entry offset index
;
gobios:
        lhld    wbvect+1 ;get cp/m warm boot vector
        dad     d        ;add in vector table offset
        pchl             ;go to table entry
;
;
;***************************************************
;                                                  *
;direct bios access routines for pl/i-80 calls     *
;                                                  *
;***************************************************
;
;
;
; routine to enter bios cold boot to completely
;reload cp/m
cboot:
        lxi     d,cbooto  ;get offset
        jmp     gobios    ;on our way
;
;
; routine to enter bios warm boot to reload
; ccp and bdos
;
wboot:
        lxi     d,wbooto  ;get offset
        jmp     gobios    ;go reload
;
;
; routine to return console status byte
; return byte value to stack
;
cstat:
        lxi     d,cstato  ;status offset
        jmp     gobios    ;return bit(8) in (a)
;                         ;from bios
;
; routine to get character from the console
; for input
;
conin:
        lxi     d,conino   ;conin offset
;
```

(continued next page)

```
retichr:                ;entry pt for 1 character
                        ;return to pl/1-80 on stack
        call    gobios  ;go get the status to (a)
        pop     h       ;return address
        push    psw     ;character to stack
        inx     sp      ;delete flags
        mvi     a,1     ;character length is 1
        pchl            ;back to calling routine
;
;
; routine to output one character to the console
;
conout:
        call    getp1   ;get single output char to (c)
        lxi     d,conouto;console output offset
        jmp     gobios  ;return to pl/1 direct from bios
;
;
; routine to output one character to the list device
;
list:
        call    getp1   ;get single output char to (c)
        lxi     d,listo ;list output offset
        jmp     gobios  ;return to pl/i direct from bios


;
;
; output routine to send one character to the punch
;
punch:
        call    getp1   ;get single output char to (c)
        lxi     d,puncho;punch output offset
        jmp     gobios  ;return to pl/i direct
                        ;from bios
;
;
; routine to get character from the reader for input
;
reader:
        lxi     d,readero ;reader input offset
        jmp     retichr ;let common code do rest
;
;
; routine to get list device status
;
lstat:
        lxi     d,lstato ;list status offset
        jmp     gobios  ;return bit(8) status
                        ;in (a) from bios
;
;
; home selected disk entry point
;
home:
        lxi     d,homeo ;home offset
        jmp     gobios  ;return direct from bios
;
;
; entry point to select disk and return parameter
; table pointer for selected drive in (hl)
;
seldsk:
        call    getp1   ;get single byte drv select byte
        lxi     d,seldsko ;select disk offset
        jmp     gobios  ;return addr pointer
                        ;in (hl) from bios
;
;
; routine to send double byte track number to bios
;
settrk:
        call    getp2   ;get track number to (bc)
        lxi     d,settrko ;set track offset
        jmp     gobios  ;return through bios
;
;
; routine to send double byte sector number to bios
;
setsec:
        call    getp2   ;get sector number to (bc)
        lxi     d,setseco ;set sector offset
        jmp     gobios  ;return through bios
;
;
; routine to send data buffer pointer to bios
;
setdma:
        call    getp2   ;get dma address to (bc)
        lxi     d,setdmao ;setdma offset
        jmp     gobios  ;return through bios
;
;
; sector translate routine.
;
sectran:
        call    getp2   ;get logsec to (bc)
        lxi     d,strano ;sectran offset
        jmp     gobios  ;return through bios
;
;
; routine to read sector of 128 bytes
;
read:
        lxi     d,reado ;read sector offset
        jmp     gobios  ;return error status
                        ;through bios
;
;
; write sector routine. entry parameter of write type
;
write:
        call    getp1   ;get write type to (c)
        lxi     d,writeo ;write sector offset
        jmp     gobios  ;return error status
                        ;through bios
;
;
;+++...end of file
```

Here I will end this section of the article; next month a sequel will go into more detail on PL/I. You'll probably want to have this part on hand for reference when you read Part 2.

Tips & Techniques

This month, our Tip again comes from Kelly Smith.

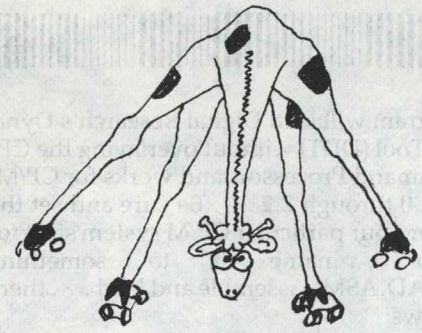
From time to time, I need to do some 'special tricks' within CP/M itself to modify its operation to do something other than what Digital Research had intended. If you attempt to load DDT 'normally', it 'clobbers' the Console Command Processor by overlaying it with DDT . . . making modification (using DDT) impossible.

This program will load Digital Research's Dynamic Debugging Tool (DDT) without overlaying the CP/M Console Command Processor, and works for CP/M version 1.4 and 2.0 through 2.2 . . . Be sure and set the equate MSIZE for your particular CP/M system size, for proper operation . . .name it to something like 'DDTLOAD.ASM', assemble and load . . . then execute it as follows:

A»DDTLOAD«cr»

DDT will now load into memory 'normally',but will be relocated BELOW the CCP. Here's the software to do it:

```
true    equ     -1          ; define true
false   equ     not true    ; define false
cpm14   equ     false       ; true if CP/M vers  1.4
cpm22   equ     true        ; true if CP/M vers  2.2
;
msize   equ     56          ; must be set for YOUR CP/M system size
;
base    equ     0           ; CP/M system base address
;
        if      cpm14
cbase   equ     base+2900h              ; base of 16k CP/M 1.4 CCP
bias    equ     (msize-16)*1024         ; offset for system
nbase   equ     cbase+bias-10h          ; new base address
        endif
;
        if      cpm22
cbase   equ     base+3400h              ; base of 20k CP/M 2.X CCP
bias    equ     (msize-20)*1024         ; offset for system
nbase   equ     cbase+bias-10h          ; new base address
        endif
;
        org     base+100h
;
        1hld    base+6      ; move old BDOS address to new BDOS entry
        shld    nbase+1     ; swap new base address into original base
        shld    base+6
        mvi     m,jmp       ; make 'jump' opcode to new base address
        lxi     h,'3D'-'0'              ; create DDT command in
                                        ; console input buffer
        shld    cbase+bias+7
        lxi     h,'DT'
        shld    cbase+bias+9
        xra     a           ; make 'EOL' as null for message end
        sta     cbase+bias+11
        mvi     a,8         ; reset command pointer
        sta     cbase+bias+136
        lda     base+4      ; select current drive
        mov     c,a
        jmp     cbase+bias  ; all parameters set, load DDT
        end
```



New Versions

The Boss Financial Accounting System
Version 1.06
This version repairs a bug concerning the printing of accounts receivable statements. Because of an incorrect date check, Version 1.04 would not print a statement to a customer if that customer was sent a statement from a previous period. Version 1.06 corrects the problem.

Owners of version 1.04 can manually clear the "Statement Sent" field of each Customer/Vendor record to allow the printing of the statements to proceed correctly.

Lifeboat CP/M for Micropolis Models I and II
Version 2.20B
This release fixes a bug in the COPY.COM program which caused the last track not to be formatted or copied. The corrected version of COPY is 5.3.

Any disk with this problem can be fixed by first using this version of COPY to format a disk fully, and then either using the M for Most option or PIP.COM to copy the files onto the newly formatted disk.

GLector
SELECTOR III-C2 Version
Version 2.02
Make the following changes and additions to the GLPERIOD.BAS program to correct the closing of a year, and the proper use of a non-calendar fiscal year. Then recompile GLPERIOD.BAS. Change the %CHAIN CODE SIZE to at least 13606 in SELECTOR.BAS. This is the last line in the program. Then recompile SELECTOR.BAS.

These changes bring GLector to the current version 2.02.

SELECTOR.BAS

```
%CHAIN 80, 13606, 400, 2030
```

GLPERIOD.BAS

```
4015  MON$=MID$(MONTH$,(MM%-1)*3+1,3)
```

(MONTH$ replaces FMONTH$.)

In the section beginning with the line label 4035

```
.
.
.
     IF INCOME%=0 THEN \          <---(line added)
        FOR I%=T.L%+1 TP TA%:\    <---(':\' added)
            LMBAL(I%)=0:\         <---(':\ added)
        NEXT
     GOTO 4050
```

In the section beginning with line label 7000

```
.
.
.
     OPEN BASE.NAME$+"1" RECL GLMON.REC.LEN% AS 1
     FOR I%=1 TO TA%
GOSUB 19100                      <---(added)
LMBAL(I%)=LYBAL(I%)
READ #1,I%;LYBAL(I%),DUMMY%
IF CLASS%>3 THEN LMBAL(I%)=0  <---(added)
PRINT #1,I%;LYBAL(I%),LMBAL(I%)
NEXT
```

Money Maestro
Version 1.1
This new release implements check printing, and deals with some bugs.

The most important bug correction in this version takes care of a miscalculation of the memory available for payees and categories, primarily affecting versions running with 48K RAM.

Nevada COBOL
Version 2.0
This version incorporates the following standard features:

1-COPY statement for library handling.
2-CALL...USING...CANCEL...EXIT PROGRAM...LINKAGE SECTION with dynamic loading of up to five active called programs at any one time and chaining to new main programs.
3-PERFORM..THRU...TIMES...UNTIL...paragraph or section names.
4-IF...NEXT SENTENCE...ELSE...NEXT SENTENCE.
5-Compound conditionals AND/OR <=> with NOT.
6-GO TO...DEPENDING ON...
7-All figurative constants including HIGH-VALUES, LOW-VALUES, and ALL.
8-All PICTURE strings and BLANK WHEN ZERO, JUSTIFIED RIGHT, SYNC, VALUE, REDEFINES, OCCURS...TIMES, USAGE COMP, COMP-3, DISPLAY.
9-120 character non-numeric literals.
10-Unique easily understood English language diagnostic error messages contained in a user changeable file.
11-CURRENCY SIGN IS ...and DECIMAL-POINT IS COMMA.
12-Interactive ACCEPT/DISPLAY...series...WITH NO ADVANCING...UNIT...
13-RELATIVE (random) access files.
14-Sequential files both fixed and variable length.
15-Choice of Display, 16-bit Binary or Packed Decimal (COMP-3) data types with up to 18-digit accuracy.

16-INSPECT...TALLYING...FOR REPLACING...BY ALL LEADING/FIRST CHARACTERS... BEFORE/AFTER INITIAL...
17-ADD, SUBTRACT, MULTIPLY, DIVIDE, GIVING, ROUNDED, ON SIZE ERROR.
19-MOVE...TO... series.
20-Default drive assignment is now standard.
21-LABEL RECORD IS OMITTED.

The bugs listed below have been corrected in this version:

1-Comparisons involving two negative numbers now are handled correctly.
2-File names with a blank File type are now treated properly.
3-A BOUNDARY ERROR message is now issued when a program falls through the last paragraph in a program.
4-Tabs in a source file are expanded by the compiler.
5- The RELATIVE KEY for a Random file can now be any data-type, i.e. Binary, Packed Decimal or ASCII Display up to 7 digits.
6-The RUN time package and the Compiler now look for a CTL-C from the console less frequently than rev 1.404, thus speeding up compile and run times.
7-Math and data conversion routines involving three decimal places have been corrected.

Postmaster

Version 3.4

These enhancements have been made to Postmaster with the new version:

1-Up to ten extraction keys may be defined and processed in a single pass, as opposed to the single key extraction formerly possible. Full source code for the extract module is provided.
2-Up to 25 cities may be defined and referenced by single letter codes. This feature can greatly reduce key entry time. The codes are saved on disk when data entry is suspended, so that they may be consulted when new information is entered later on.
3-A 'fast-list' command has been added to PMEDIT, displaying each record in compressed format on a single line. The new command makes it much easier to locate individual records when paging through a file.
4-PMLABL, PMTYPE, and PMENVE automatically produce the correct 5+4 output format for nine-digit Zip codes whenever all nine character positions in the Zip field contain numeric information only.

S-BASIC

Version 5.4

This information should be added to the S-BASIC documentation:

If the system is being implemented on a CP/M compatible system other than CDOS (Cromemco), the two '.CDS' files are not used. If your system is CDOS, then RENAME the two '.CDS' files to '.REL' on your working copies.

Note that only the FAC.BAS program can be functionally compiled as is. See your application notes for the use of bound-in assembly routines as demonstrated using the other program files.

SELECTOR IV

Version 2.14A

A change in this package dates from July 20, 1981. If the last record in a file is deleted and no other 'Key field' modification takes place during a file update session, the '.KEY' file will not be properly updated. The changes below in UPDATE.BAS (and GLTRANS4.BAS if you have the GLector IV system) correct this. The change in BATCH.BAS corrects a similar situation.

The change in SELECTOR.BAS is required because of the slightly larger code size of BATCH.BAS. The other changes in GLPERIOD.BAS and GLTRANS4.BAS (of GLector IV insure correct operation when transactions involving 6 ledger accounts are processed. After making the changes, recompile the programs to be sure that COMMON.BAS, SCREENS.BAS, all DATE?.BAS files, and GETFILE.BAS are on the same disk drive as the program being compiled.

Changed sections, unless otherwise noted, appear in bold.

UPDATE.BAS

```
28000
     IF I%=3 THEN MENU$=" B":\
             XXX$=""
     IF LIM% THEN MENU$=" B "+FIL.NAME$
```

Below three lines added 7/20/81:

```
     IF SAVE.DELETE.RP AND KEY.COUNT%=0 THEN \
             KEY$(1)="END":\
             KEY.COUNT%=1
IF KEY.COUNT% THEN DUMMY%=FN.WRITE.KEY%
```

BATCH.BAS

```
DEF  FN.DELETE.EXECUTE%
     IF OPEN%(20) THEN CLOSE 20
     OPEN "DELETE" AS 20
     OPEN%(20)=-1
     IF END #20 THEN 0.59
     END.OF.FILE%=0
     WHILE NOT END.OF.FILE%
          READ #20;FILE%,R.P
          READ #FILE%,TA(FILE%-1);LINE DREC$
          PRINT USING "&";#FILE%,TA(FILE%-1);CHR$(26)
          TA(FILE%-1)=TA(FILE%-1)-1
```

Below line changed 7/20/81:

```
          IF  R.P<=TA(FILE%-1) THEN PRINT USING
                                    "&";#FILE%,R.P;DREC$
```

(This is all one line.)

(continued next page)

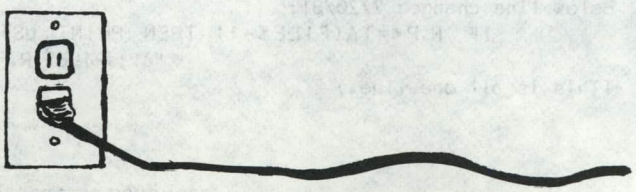
```
                    SELECTOR.BAS
%CHAIN 60, 17550, 0, 3520 <-- (17550 as of 7/20/81)

                    GLTRANS4.BAS
     PAGE%=0
     PAGE%=1 <--(added 7/20/81)
     ITEM.LINE%=1
---------

DEF  FN.CFIELD$
     DUMMY$=FN.NEXT.FIELD$
               ('+"    "' below added 7/20/81)
     IF LEN(CFIELD$(GFIELD%)) THEN \
          PRINT USING"&";CFIELD$(GFIELD%)+"    "\
     ELSE DUMMY$=FN.US$:\
---------

%INCLUDE DATE4

Delete all of FN.MASK%
---------

71110 IF MATCH(STR$(GFIELD%),"6 8 10 12 14 16",1)
                                        AND \
          VAL(CFIELD$(GFIELD%))=0\
               OR GFIELD%>16 THEN GFIELD%=1:RETURN
--------

     DUMMY%=FN.MASK%(CFIELD%) <--deleted 7/20/81
     GLTRANS$=DRIVE$+BASE.NAME$+"DAT"
---------

28000  IF LIM% THEN MENU$=" M "+LEFT$(GLTRANS$,LEN
                              (GLTRANS$)-4)+" "+\
          KDRIVE$+" "+STR$(I%)
Below three lines added 7/20/81:
IF KEY.COUNT%=0 AND SAVE.DELETE.RP THEN \
          KEY$(1)="END":\
          KEY.COUNT%=1
IF KEY.COUNT% THEN DUMMY%=FN.WRITE.KEY%
---------

                    GLPERIOD.BAS
DIM ACT$(7),AMNT(7) REM <--7 versus 6, 7/20/81
```

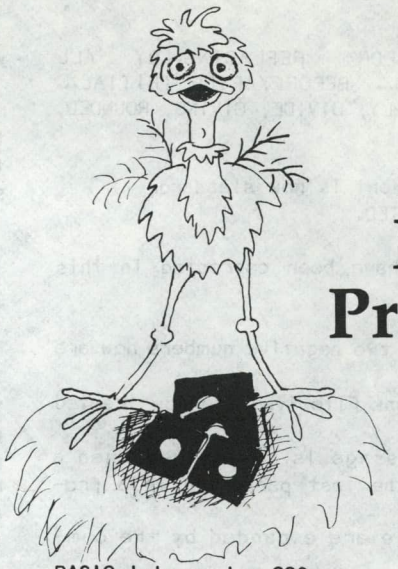
T/MAKER II

Version 2.2.3

In this new version the Files statement should now work for CP/M 1.4 as well as 2.x. Previously, it did not work properly for CP/M 1.4.

A bug in the Combine function caused a data row in the table in memory to be ignored if the first character on the line appeared after column 6. Correct processing should be that the row is ignored only if the first character appears after column 7. This problem has now been fixed.





New Products

BASIC Interpreter-280

by Control-C Software

This BASIC Interpreter runs under CP/M 2.2; it is designed to accompany Business BASIC II (Basic Four Corp.) and requires a 64K RAM CP/M system. Utilities include renumbering, directory management, and an installation utility for configuration to particular terminals.

CP/M MITS Q70 Driver

by The Software Store

This driver emulates the Diablo 1620/1650/630 printers using the MITS Q70 printer and the CP/M operating system. The driver supports proportional spacing and full use of word processing systems. An entry point for the CP/M List Output Status routine is provided.

The following Diablo control codes are supported by the driver: Backspace, Carriage Return, Space, Forward Print On, Backward Print On, Set Left Margin, Reverse 1/2 Linefeed, Linefeed, Define VMI (Vertical Motion Index), Define HMI (Horizontal Motion Index), Set Form Length, 1/2 Linefeed, Reverse Linefeed, Absolute Horizontal Tab, Absolute Vertical Tab, and Form Feed. An additional code sequence for setting the form length is provided because this function is accomplished by the Diablo with a switch.

Floating Point FORTH

by Timin Engineering Company

The 16- and 32-bit integer capability of FIG FORTH is retained, but a floating point mode is added. These single precision floating point numbers have approximately seven significant decimal digits. Addition, subtraction, negation, multiplication, division, integer conversion, and comparison are supported. Run time error messages warn the user of exceptional conditions and allow processing to continue at the user's option.

Floating point will run on Z80/8080/8085 hardware, requires a minimum memory size of 28K.

OASIS Pascal

by Phase One Systems

Modelled on UCSD Pascal, the OASIS version permits

separate compilation of routines. Construction of procedure libraries is provided by new 'module', 'export block', and 'import block' declarations. Separately compiled modules are assembled into linking loader formats and combined with other modules to form libraries of routines and global variables. Without restrictions on the order of declarations, any constants, types, variables, and routine definitions may be grouped together for better reading and modularity.

Variables can be initialized and explicit type conversion is permitted. In addition, syntax for structured constants has been added. Integer literals may be represented in a radix other than decimal.

PLAN80

by Business Planning Systems

This financial applications package is written in Pascal. It is designed for planning, forecasts, budgets, and analyses; it produces printed or screen reports. Problems are described in rows and columns, allowing the user to input data values, specify calculations, and communicate results between PLAN80 applications. Results may be graphed or displayed in numeric form.

If a user employs 13 columns s/he can create up to 235 rows. If there are 100 rows, 40 columns can be used. The manual supplies a table showing the limitations on the number of rows and columns.

PLAN80 requires and 8080 or Z80 CPU, CP/M, 56K RAM, a console with clear screen and cursor control features. Two disk drives with a minimum of 100K each are recommended though not required.

Stiff Upper Lisp

by Harry Tennant

This interactive computer language includes an editor, trace package, break package, single step facility, and on-line help facility (user extensible), a formatter, a spelling correction function, a history package to maintain a history of the user/computer interaction. This package permits the user to edit and redo inputs, to save or manipulate system outputs. An initialization file is automatically loaded to customize the environment and to define autoloading functions. Function definitions are loaded into primary memory only when executed, so no space is used unless needed. Context switching is performed through the use of splicing read macros.

Stiff Upper Lisp stores more than 120 compiled functions into less than 14K bytes. It includes a library of functions written in Lisp.

Closure functions may be defined to encapsulate programs and data, to help in writing generator functions and coroutines, to have automatic tail recursion elimination so that recursive functions can be written as efficiently as interactive functions. Closure functions may be defined to enable building tree structured binding environments, providing modularity through class inheritance.

Stiff Upper Lisp requires CP/M or TRSDOS, 32K to 64K RAM (at least 48K is recommended), a minimum disk capacity per drive of 75K.

BUGS

Lifeboat CP/M For TRS-80 Model II

Version 2.25A

When one runs the COPY program, and copies from A: to B: and the copy is completed, the first choice of the menu states that a carriage return will copy again using the same parameters.

Actually, the program reverts back to the default of copying from A: to B:.

Lifeboat CP/M 28 For TRS-80 Model II

Version 2.25A

The GETFILE of this version is not compatible with past versions.

PMATE

Version 2.06

PMATE will bomb when run under CP/M 2.25A on the TRS Model II if ERASE TO END OF LINE, LINE INSERT and LINE DELETE are implemented in the CNF file. PMATE works fine when these functions are removed.

Unlock

Version 1.3

If one is logged onto drive A: Unlock's FILENAME.BAS will read FILENAME.BAS from drive A:. Unlock A:FILENAME.BAS will read FILENAME.BAS from drive B:. B:FILENAME.BAS will be read from C:, etc.

XASM-09

Version 1.05

Negative displacements in Load Effective Address instructions are not calculated properly. In computing displacements between -32 and -17, and +16 and +31, the program calculates a 5-bit displacement rather than the proper 8-bit displacement.

The following patch fixes it:

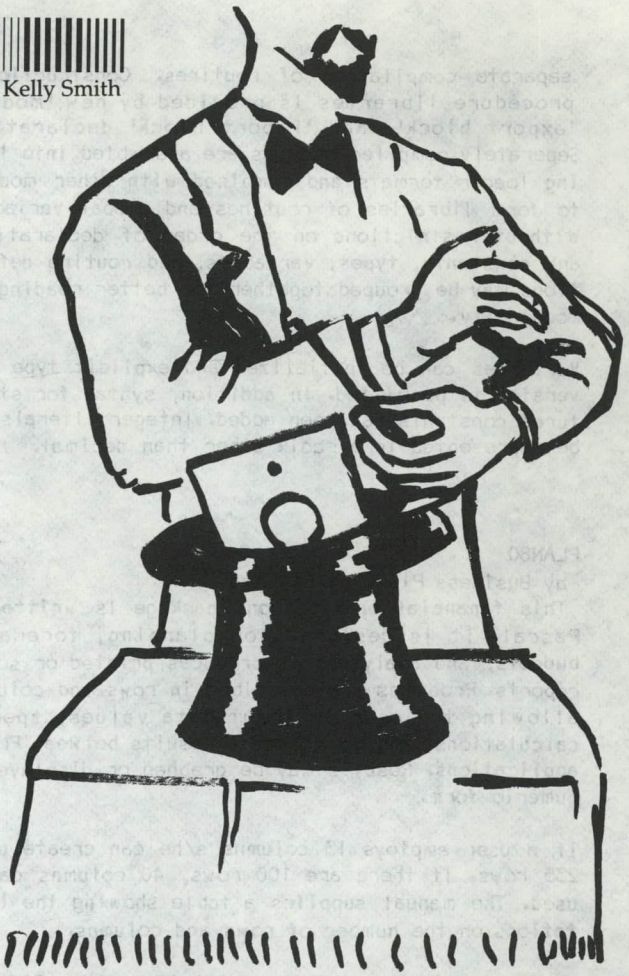
At locations 215c, 2163, and 2165 replace the E0 with an F0.

MicroPro Software On the Apple

Except for the special WordStar for the Apple, no MicroPro International Corporation software will run on the Apple. MicroPro is working on custom versions of their products for the Apple.

A Software Trick

by Kelly Smith



In my travels through the software written by others (articles, disassemblies, etc.), I occasionally find some "tricks" incorporated to either optimize the storage requirements of the code (typically ROM based) or to attempt to confuse disassembly... although this is not a recommended routine, I think you may find it of interest.

The LXI Trick

Many large mainframe computers (minis and maxis) have a SKIP instruction... but most micros are multi-byte instruction oriented, and therefore it is difficult to provide a SKIP when the instruction length is indeterminate. First, an example of "straight" coding:

```
error1:     mvi     a,1   ;   set-up error code 1
            jmp     error$handler
error2:     mvi     a,2   ;   set-up error code 2
            jmp     error$handler
error3:     mvi     a,3   ;   set-up error code 3, and fall into it
;
error$handler:            ;   all error codes come here
;
            lxi     d,error$message ;   point to error message,
                                    ;   error code in A reg.
            .
            .
            .
```

This is easy enough to understand (right?), but consider this:

```
lxib        equ     1     ;   equate first byte of LXI  b,nnnn
;
error1:     mvi     A,1   ;   set-up error code 1
            db      lxib  ;   first byte of LXI  B,nnnn
error2:     mvi     A,2   ;   set-up error code 2
            db      lxib  ;   first byte of LXI  B,nnnn
error3:     mvi     A,3   ;   set-up error code  3
            lxi     d,error$message   ;   point to error message
                                      ;   error code in A  reg.
            .
            .
            .
```

If a jump is made to ERROR1, the E Reg. is set-up, then the LXIB will be executed, the B&C Regs. will be given "garbage" code that follows it, and finally the program counter will be incremented past the next instruction... it SKIPs and falls into the eventual output routine... insidious to disassemblers! This was one of the "favorite's" at MITS (remember the Altair?).

Some People Spell Their Time Away... *Smart* People Use

MicroSpell

Spelling by Memory

1. Write masterpiece.
2. Use typewriter.
3. Search for Misspellings.
4. Find Dictionary.
5. Look up incorrect words.
6. Use dictionary.
7. Repeat steps 3 through 6 until you think you've finished.
8. Retype masterpiece.
9. Rest from eyestrain.
10. Hope you were right.

Spelling With MICROSPELL, The Spelling Corrector

1. Write masterpiece.
2. Use Word Processor.
3. Run **MicroSpell**.
4. Print Masterpiece.
5. Take the rest of the day off.

Spelling with The Competition

1. Write masterpiece.
2. Use Word Processor.
3. Run spell checker.
4. Find Dictionary.
5. Re-run word processor.
6. Find Marked word.
7. Use Dictionary.
8. Correct using the word processor.
9. Repeat steps 6 through 8 until you're finished.
10. Print masterpiece.
11. Rest from eyestrain.

The choice is yours.

For more information on **MicroSpell**, contact Dept. P at our New York headquarters: 1651 Third Ave., New York, N.Y. 10028, Phone: (212) 860-0300, TWX: 710-581-2524 LBSOFT NYK, Telex: 640693

Lifeboat Worldwide:
Lifeboat Inc., OK Bldg. 5F, 1-2-8, Shiba-Daimon, Minato-ku, Tokyo 105, Japan, Tel: 03-437-3901, Telex: 2423296 (LBJTYO)
Lifeboat Associates, Ltd., PO Box 125, London WC2H 9LU, U.K., Tel: 01-836-6921, Telex: 893709 (LBSOFTG)
Lifeboat Associates, GmbH, Schlossgartenweg 5, D-8045 Ismaning, W. Germany, Tel: 089-966-444, Telex: 5213643 (ISOFD)
Lifeboat Associates, SARL, 10 Grande Rue Charles de Gaulle, 92600 Asnieres, France, Tel: 1-733-08-04, Telex: 250303 (PUBLIC X PARIS)

Lifeboat Associates

Software with full support.

Operating Systems

Description	Version
These CP/M operating systems are available from Lifeboat Associates except where otherwise mentioned.	
Apple II w/Microsoft BASIC	**2.0B**
Cromemco System 3 8"	1.4
Cromemco System 3 8"	2.2
Datapoint 1550/2150 DD/SS	2.2
Datapoint 1550/2150 DD/DS	2.2
Durango F-85	2.23
Heath H8 with H17 Disk	1.43
H89 Heath/Zenith-Magnolia	2.2
iCOM 3812	1.41
iCOM 3712 w/Altair Console	1.41
iCOM 3712 w/IMSAI Console	**1.42**
iCOM Microfloppy(2411)	1.41
iCOM 4511/Pertec D3000 Hard	2.22
Intel MDS Single Density	2.2
Intel MDS 800/230 Double Density	2.2
MITS Altair 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
North Star SD	1.41
Mostek MDX STD Bus	2.2
Sol North Star SD	1.41
North Star SD IMSAI SIO Console	1.41
North Star SD MITS SIO Console	1.41
North Star DD	1.45
North Star SD	**2.23A**
North Star DD/QC w/Corvus	2.22
North Star DD/QD	2.23A
Ohio Scientific	2.24
Ohio Scientific C-3B	**2.24B**
Ohio Scientific C-3C' (Prime)	**2.24B**
Processor Technology Helios II	1.41
from Lifeboat TRS-80 5 1/4" (Mod I)	1.41
from Lifeboat TRS-80 Mod II	2.25A
from Cybernetics TRS-80 Mod II	2.25

Hard Disk Modules

Description	Version
Corvus Modules	**2.1**
Apple-Corvus Module	**2.1a**
KONAN Phoenix Drive	1.8
Micropolis Microdisk	1.92
Pertec D3000/iCOM 4511	1.6
Tarbell	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**

New products and **new versions** appear in boldface.

Available Memory Requirements

Currently memory requirements for a program are specified in terms of Minimum System Size. This method of measuring a computer's capacity is misleading because there is no standard relationship between the nominal system size and the actual amount of memory available for program use. Two computers which sign on with the same system size message can differ in usable memory by 10K, depending upon the complexity of the operating system implementation.

Therefore, in an effort to provide a totally standard method of specifying memory requirements, we will be referring to the "Available Memory Requirement" of a program. Generally, the Available Memory of a computer will be 6K to 8K less than the system size. To find the exact amount of Available Memory on your system, run the following program, which is written in Microsoft BASIC.

```
10 '**          PROGRAM TO FIND AVAILABLE MEMORY
20 MEMSIZ = INT((PEEK(6)+256*PEEK(7))/1024)
30 PRINT "Your Computer Has";MEMSIZ;"K Bytes of
                                    Available Memory"*
40 END
```

In this way, you can be certain if a computer program will run on your computer. Authors of the computer packages listed in Lifelines' Version List, are encouraged to inform us of the exact Available Memory Requirements of their programs so that they may be included in next month's issue.

*This line and the line above it should appear as one line.

Version List

The listed software is available from the authors, computer stores, distributors, and publishers.

These prices are given as approximations. Actual prices for these products will vary from vendor to vendor.

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

KEY

S	Standard Version	P	Processor
M	Modified Version	MR	Memory Required
OS	Operating System	$	Price

VERSION LIST

August 14, 1981

Product	S	M	OS	P	MR	$	
ACCESS-80	1.0		CP/M	8080/Z80	54K	792	
Accounts Payable/Cybernetics	3.1		CP/M	Z80	64K	500	Needs RM/COBOL
Accounts Payable/MC	**1.0**		**CP/M**	**8080/Z80**	**56K**	**600**	**CP/M 2.2**
Accounts Payable/Structured Sys	1.3B		CP/M	8080	52K	840/40	
Accounts Payable/Peachtree	7-13-81		CP/M	8080	48K	530/60	Needs OBASIC
Accounts Receivable/Cybernetics	3.1		CP/M	Z80	64K	500	Needs RM/COBOL
Accounts Receivable/MC	**1.0**		**CP/M**	**8080/Z80**	**56K**	**600**	**CP/M 2.2**
Accounts Receivable/Peachtree	7-13-81		CP/M	8080	48K	530/60	Needs OBASIC
Accounts Receivable/Structured Sys	1.4C		CP/M	8080	56K	840/40	
Address Mngemt.Sys	1.0		CP/M	8080		150	Requires 2 drives
ALDS TRSDOS	**3.4**		**TRSDOS**		**32K**	**80/35**	
ALGOL/60	4.8C		CP/M	8080	24K	250	
ANALYST	2.0		CP/M	8080	52K	250/20	Needs OBASIC, QSORT or VSORT
APL/V80 Interpreter	3.2		CP/M	Z80	48K	500	Needs APL terminal
Automated Patient History	1.2		CP/M	8080	48K	175	
BASIC-80 Compiler	5.3	5.24	CP/M	8080	48K	360/35	
BASIC-80 Interpreter	**5.21**	**5.21**	**CP/M**	**8080**	**40K**	**335/35**	**W/Vers. 4.51,5.2**
Basic Interpreter-280			**CP/M**	**8080/Z80**	**64K**	**600**	**Needs CP/M 2.2**
BASIC Utility Disk	2.0	2.0	CP/M	8080	48K	75	
BSTAM Communication System	4.5	4.5	CP/M	8080	16K	200	
BSTMS	1.2	1.2	CP/M	8080	24K	200	
BUG/uBUG Debuggers	2.03		CP/M	Z80		129/25	Not for CDOS
BDS C Compiler	1.44	1.44T	CP/M	8080	32K	150/30	
Boss Finc'l Acctg Sys	**1.06**		**CP/M**	**8080**	**54K**	**2495**	**Needs 2 drive,min. 200K ea.**
Whitesmith's C Compiler	2.0		CP/M	8080	60K	630/30	
CBASIC2	2.07P	2.17P	CP/M	8080	32K	125/20	
CBS Applications Builder	1.3		CP/M	8080	48K	395/40	CDOS version too
CIS COBOL Standard	4.3,1		CP/M	8080	48K	850/50	
CIS COBOL Compact	3.46	3.46	CP/M	8080	32K	650/50	
FORMS 1 CIS COBOL Form Generator	1.06	1.06	CP/M	8080		150/20	
FORMS 2 CIS COBOL Form Generator	1.1,6	1.16	CP/M	8080		200/20	
Interface for Mits Q70 Printer			CP/M			150	CP/M 1.41 or 2.XX
COBOL-80 Compiler	4.01A	4.01A	CP/M	8080	48K	710/35	
COBOL-80 PLUS M/SORT	4.01		CP/M	8080	48K	845/65	
CONDOR	1.10		CP/M	Z80	48K	695/35	
CREAM (Real Estate Acct'ng)	2.3		CP/M	8080	64K	250	Needs CBASIC
Crosstalk	**1.4**		**CP/M**	**Z80**		**150**	
DATASTAR Information Manager	1.101		CP/M	8080	48K	350/60	
Datebook	2.03		CP/M	8080	48K	295/30	
dBase II	**2.02A**		**CP/M**	**8080/Z80**	**48K**	**700/50**	
Dental Mngmt. System	8.7		CP/M	8080/Z80	48K	750/NA	Requires CBASIC2
DESPOOL Print Spooler	1.1A		CP/M	8080	19K	80	
DISILOG Z80 Disassembler	4.0	4.0	CP/M	Z80	24K	110	Zilog mnemonics
DISTEL Z80/8080 Disassembler	4.0		CP/M		24K	110	
EDIT Text Editor	**2.06**		**CP/M**	**Z80**		**129/25**	
EDIT-80 Text Editor	2.02		CP/M	8080		99/25	
ESQ-1	2.1A		CP/M	8080	48K	1495/50	Needs CBASIC2
FABS	2.4A		CP/M	8080	32K	195/25	
FILETRAN		1.2	TRSDOS		32K	99/20	1-way TRS-80 Mod I, TRSDOS-CP/M
FILETRAN		1.4	CP/M		32K	149/20	2-way TRS-80 Mod 1, TRSDOS & CP/M
FILETRAN	1.5		CP/M		32K	99/20	1-way TRS-80 Mod II, TRSDOS -CP/M
Financial Modeling System	2.0		CP/M		48K	300	
Floating Point FORTH	**2**		**CP/M**	**8080/Z80**	**28K**	**195**	
Floating Point FORTH	**3**		**CP/M**	**8080/Z80**	**28K**	**335**	
FORTRAN-80 Compiler	**3.43**	**3.43**	**CP/M**	**8080**	**36K**	**435/35**	
FORTRAN TRS	**3.42**		**TRSDOS**			**80/35**	
FPL	2.0	2.0	CP/M	8080	48K	695/30	
General Ledger/Cybernetics	1.3C		CP/M	Z80	48K	500	Needs RM/COBOL
General Ledger/MC	**1.0**		**CP/M**	**8080/Z80**	**56K**	**600/10**	**CPM 2.2 or MPM**
General Ledger/Peachtree	7-13-81		CP/M	8080	48K	530/60	Needs OBASIC
General Ledger/Structured Sys	1.4C		CP/M	8080	52K	840/40	
GLECTOR Accounting System	**2.0**		**CP/M**	**8080**	**56K**	**350/25**	**Use w/CBASIC2, Selector III C-2**
HDBS	1.04		CP/M	+	52K	300	
Lifeboat IBM/CPM	1.1		CP/M	8080		175	
Integrated Acctg Sys/Gen'l Ledger			**CP/M**	**8080**	**48K**	**150/25**	**Needed for pkgs below**
Integrated Acctg Sys/Accts Pyble			**CP/M**	**8080**	**48K**	**250/25**	
Integrated Acctg Sys/Accts Recvble			**CP/M**	**8080**	**48K**	**250/25**	
Integrated Acctg Sys/Payroll			**CP/M**	**8080**	**48K**	**250/25**	
Interchange			**CP/M**	**8080**	**32K**	**59.95/10**	**CP/M2, Mtchmker for N.S. users**
Inventory/MicroConsultants	**5.3**		**CP/M**	**8080/Z80**	**56K**	**995**	**Needs CP/M 2.2**
Inventory/Peachtree	7-13-81		CP/M	8080	48K	530/60	Needs OBASIC
Inventory/Structured Sys	1.0C		CP/M	8080	52K	640/40	
Job Cost Control System/MC	**1.0**		**CP/M**	**8080/Z80**	**56K**	**600**	**Requires CP/M 2.2**
JRT Pascal	1.4		CP/M	8080	56K	225/25	
LETTERIGHT Text Editor	1.1B		CP/M	8080	52K	200/25	
MAC	2.0A		CP/M	8080	20K	120/25	
MACRO-80 MACRO Assembler Package	**3.43**	**3.43**	**CP/M**	**8080/Z80**		**159/25**	
Magic Wand	1.11		CP/M	8080	32K	395/40	
MAGSAM III	4.2		CP/M	8080	32K	145/25	For CBASIC/MBASIC
MAGSAM IV	1.1		CP/M	8080	32K	295/25	Needs CBASIC
MAILING ADDRESS Mail List System	7-13-81		CP/M	8080	48K	530/60	Needs OBASIC
Mail Merge	2.26		CP/M	8080		150/25	Needs same version Wordstar
Matchmaker			**CP/M**	**8080**	**32K**	**110/5**	
MDBS	1.04		CP/M	+	48K	900/35	
MDBS-DRS	1.02		CP/M	+	52K	300	
MDBS-QRS	1.0		CP/M	+	52K	300	
MDBS-RTL	1.0		CP/M	+	52K	300	

The listed software is available from the authors, computer stores, distributors, and publishers.

VERSION LIST

Product	S	M	OS	P	MR	$	Notes
Microspell	4.2		CP/M	8080	48K	249	
Medical Mngemt. System	8.7		CP/M	8080/Z80	48K	750/NA	Needs CBASIC2
Microstat	2.0a		CP/M	8080/Z80	48K	250/NA	Needs BASIC-80,5.03 or above
Mince	**2.5**		**CP/M**	**8080/Z80**	**56K**	**125/20**	
Mini-Wrehse Mngmt Sys	5.5		CP/M	8080	48K	650	
MP/M Operating System	1.1		MP/M	8080	32K	300/50	For Intel MDS 800
MSORT	4.01		CP/M	8080	48K	160/20	Needs COBOL-80
Mu LISP-80	**2.10**		**CP/M**	**8080**	**24K**	**210/25**	
Mu SIMP/Mu MATH Package	**2.10**		**CP/M**	**8080**	**48K**	**260/30**	**muMATH 80**
NAD Mail List System	3.0D		CP/M	8080	48K	115/25	
Nevada COBOL	**2.0**	**2.0**	**CP/M**	**8080**	**32K**	**149/25**	
r Entry w/Inv			CP/M	Z80		500	Needs RM/COBOL
PAS-3 Medical	1.76		CP/M	8080	56K	995/25	Needs CBASIC2
PAS-3 Dental	1.62		CP/M	8080	56K	995/25	See above
PASM Assembler	1.02		CP/M	Z80		129/25	
Pascal/M	3.2		CP/M	8080	56K	175/25	Also for CDOS
Pascal/MT Compiler	3.2		CP/M	8080	32K	250/30	
Pascal/MT +	5.25		CP/M	8080	52K	500/30	For Z80 too
Pascal/Z Compiler	**4.0**		**CP/M**	**Z80**	**56K**	**395/25**	
Payroll/Cybernetics			CP/M	Z80		500	Need RM/COBOL
Payroll/Peachtree	7-13-81		CP/M	8080	48K	530/60	Needs OBASIC
Payroll/Structured Sys	1.0E		CP/M	8080	56K	840/40	
PL/I-80	1.3		CP/M	8080	48K	500	
Plan-80	**2.0**		**CP/M**	**8080/Z80**	**56K**	**295/40**	
PLINK II	1.08		CP/M	Z80		350	
PLINK Linking Loader	3.25P		CP/M	Z80		129/25	
PMATE	2.06		CP/M	8080	32K	195	
POSTMASTER Mail List System	**3.4**	**3.4**	**CP/M**	**8080**	**48K**	**150/20**	**Needs CBASIC**
Professional Time Acctg	3.11a		CP/M	8080	48K	595/30	Needs CBASIC2
Property Manager	7-13-81		CP/M	8080	48K	925/60	Needs OBASIC
PSORT	1.1		CP/M	8080		100	
QSORT Sort Program	2.0		CP/M	8080	48K	100	
RAID	4.7.3A	4.7.3	CP/M	8080	28K	250/25	
Real Est Aquisition Prog	2.1		CP/M	8080	56K	500	Needs CBASIC
Remote	**3.01**		**CP/M**	**Z80**		**150**	
Residential Prop Mngemt Sys	1.0		CP/M	Z80	48K	650	
Lifeboat RECLAIM Verification Prog	2.1		CP/M	8080		80	
SBASIC	5.4		CP/M	8080		295/35	
Scribble	**1.2**		**CP/M**	**8080/Z80**	**56K**	**175/25**	
SELECTOR III-C2 Data Manager	3.24		CP/M	8080	48K	295/25	Needs CBASIC
SELECTOR IV	**2.14a**		**CP/M**	**8080**	**52K**	**550/35**	**Needs CBASIC**
Shortax	1.2		CP/M	Z80	48K	500/15	TRSDOS,MDOS too,needs BASIC-80 5.0
SID Symbolic Debugger	1.4		CP/M	8080		120/25	N/A-Superbr'n
SMAL/80 Programming Sys	3.0		CP/M	8080		75/25	For CP/M 1.x
Spellguard	**1.0**		**CP/M**	**8080/Z80**	**32K**	**295/20**	**Needs Word Processing Program**
STATPAK	1.2	1.2	CP/M	8080		495/30	Needs BASIC-80 4.2 or above
STRING BIT	1.02	1.02	CP/M	8080		75/25	
STRING/80 bit	1.22		CP/M	8080		95/25	
STRING/80 bit Source	1.22		CP/M	8080		295	
Supersort I Sort Package	1.5		CP/M	8080		225/40	Max.record = 4096 bytes
T/MAKER II Data Calculator	**2.2.3**		**CP/M**	**8080/Z80**	**48K**	**275/25**	
T/MAKER II Demo	2.2.1		CP/M	8080	48K	50	
TEX Text Formatter	1.1		CP/M	8080	36K	105/50	
TEXTWRITER-III	3.6	3.6	CP/M	8080	32K	125/20	
TINY C Interpreter	800102C		CP/M	8080		105/50	
TINY C II Compiler	800201		CP/M	8080		250/50	
TRS-80 Customization Disk	1.3		CP/M	8080		75	
ULTRASORT II	4.1A		CP/M	8080	48K	195/25	
Lifeboat Unlock	1.3		CP/M	8080		95	Use w/BASIC-80 5.2 or above
Visicalc	1.37		Apple	8080	32K	150	
Wordindex	3.0		CP/M	8080	48K	195	Needs WordStar
WordMaster	1.07A		CP/M	8080	40K	145/40	
WordStar	**3.0**		**CP/M**	**8080**	**48K**	**445/60**	
WordStar w/MailMerge	**3.0**		**CP/M**	**8080**	**48K**	**575/85**	
XASM-05 Cross Assembler	1.04		CP/M	8080	48K	195/25	
XASM-09 Cross Assembler	1.05		CP/M	8080	48K	200/25	
XASM-51 Cross Assembler	1.07		CP/M	8080	48K	200/25	
XASM-F8 Cross Assembler	1.03		CP/M	8080	48K	200/25	
XASM-400 Cross Assembler	1.02		CP/M	8080	48K	200/25	
XASM-18 Cross Assembler	1.40		CP/M	8080	48K	200/25	
XASM-48 Cross Assembler	1.60		CP/M	8080	48K	200/25	
XASM-65 Cross Assembler	1.96		CP/M	8080	48K	200/25	
XASM-68 Cross Assembler	1.99		CP/M	8080	48K	200/25	
XMACRO-86 Cross Assembler	3.40		CP/M	8080	48K	275/25	
XYBASIC Interpreter Extended	2.11•		CP/M	8080		450/25	
XYBASIC Interpreter Extended CP/M	2.11		CP/M	8080		550/25	
XYBASIC Interpreter Extended COMP	2.0		CP/M	8080		450/25	
XYBASIC Interpreter Extended ROM	2.1		CP/M	8080		450/25	
XYBASIC Interpreter Integer	1.7		CP/M	8080		350/25	
XYBASIC Interpreter Integer COMP	2.0		CP/M	8080		350/25	
XYBASIC Interpreter Integer ROM	1.7		CP/M	8080		350/25	
Z80 Development Package	3.5		CP/M	Z80		130	N/A Suprbrn,Magnolia,mod CP/M
ZDM/ZDMZ Debugger	1.2/2.0		CP/M	Z80		45	For Micropolis,N'Star,Apple,IBM 8"
ZDMZ Debugger	2.0		CP/M	Z80		45	See note above
ZDT Z80 Debugger	1.41	1.41	CP/M	Z80		50	N/A Superbr'n,mod CP/M
ZSID Z80 Debugger	1.4A		CP/M	Z80		130	See note above.

These prices are given as approximations. Actual prices for these products will vary from vendor to vendor.
New products and **new versions** are listed in boldface.
+ These products are available for Z80 or 8080, in the following host languages: BASCOM, COBOL-80, FORTRAN-80, PASCAL/M, PASCAL/Z, CIS COBOL, CBASIC, PL/1, and BASIC-80 5.xx.

LIFELINES®

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

NEW YORK N. Y.

FIRST CLASS MAIL
U. S. POSTAGE
PAID
Permit No. 416

FIRST CLASS MAIL