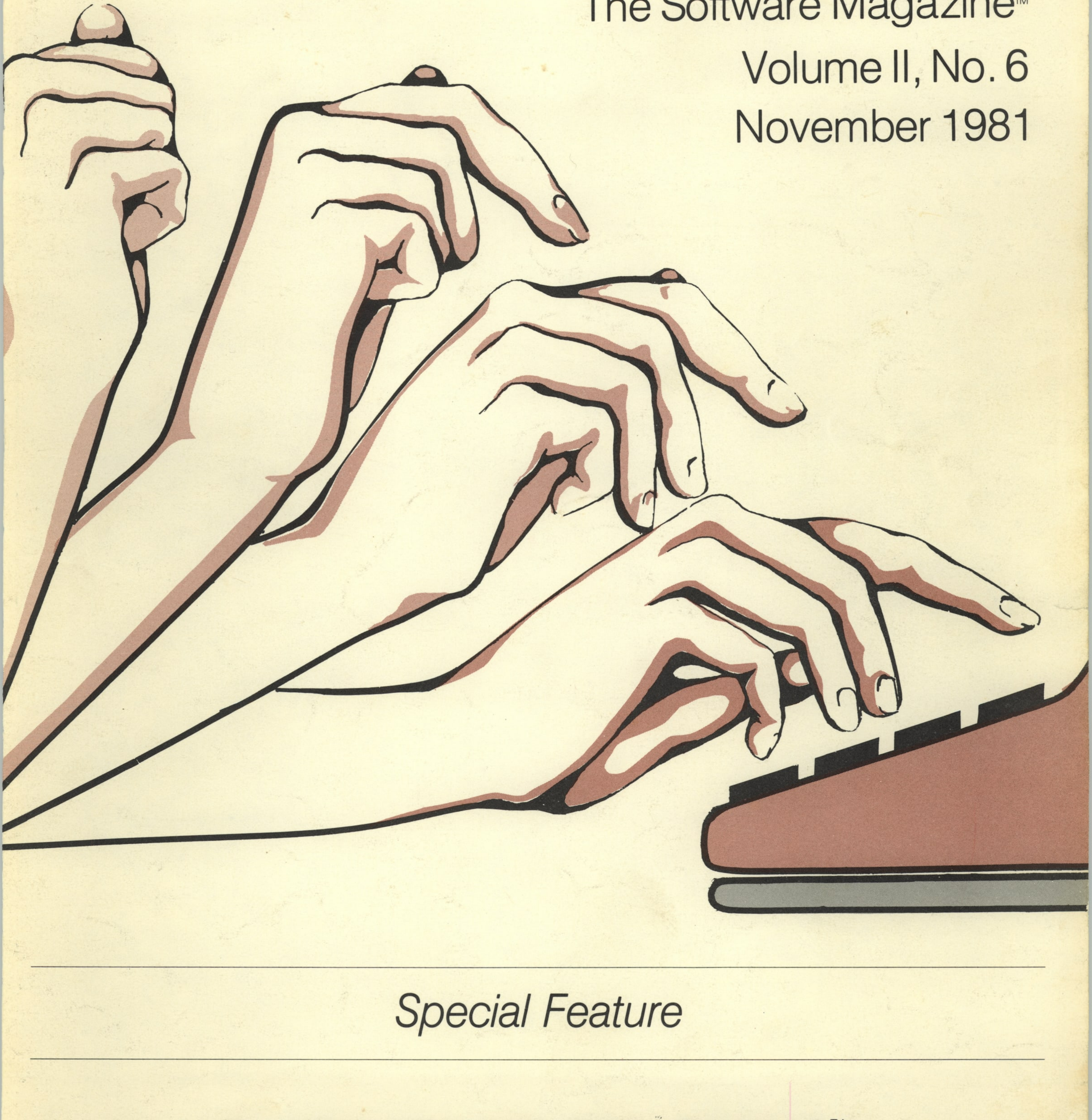
LIFELINES®

The Software Magazine™

Volume II, No. 6
November 1981



Special Feature

Ten New Volumes From CPMUG™!

LIFELINES®

Editor-in-Chief: Edward H. Currie
Managing Editor: Jane V. Mellin
Production Assistant: K. Gartner
Administrative Assistant: Susan M. Sawyer

Volume II No. 6 November

Contents

Lifelines, Volume II, Number 6. Published monthly by Lifelines Publishing Corporation, 1651 Third Ave., New York, N.Y. 10028. Telephone: (212) 722-1700. 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 when destined for any other country. All checks should be made payable to Lifelines Publishing Corporation. Foreign checks must be in U.S. dollars, drawn on a U.S. bank; checks, money orders, VISA, and MasterCard are acceptable. All orders must be pre-paid. Please send all correspondence to the Publisher at the above address. Postmaster, please send change of address to Lifelines Publishing Corporation, 1651 Third Ave., New York, N.Y. 10028. Application to mail at second class postage pending at New York, N.Y.

Editorial Comments

A Call for Standards

Last month we touched briefly on the historical aspects of what we have chosen to call the "Age of Altair". This month the topic to be reviewed is that of "Standards" in the microcomputer world.

The first microcomputer standard began as the result of H. Edward Roberts' definition of the Altair bus. This bus, which was never copyrighted, rapidly gained widespread use and soon after, a major campaign was launched by Carl Warren and others to rename it with the generic, non-proprietary name, S100. This effort was based largely on the belief that the name would place the design securely in the public domain and that ultimately it would evolve to a true industry standard. Soon thereafter, through the efforts of George Morrow, Mark Garetz, Steve Adelman and others, the process was started to develop an IEEE specification for the S100 bus and today the proposals are about to be adopted as a fully sanctioned standard.

The world of microcomputer software has not experienced anything quite like its hardware counterpart but rather has tended for the most part to evolve *de facto* standards. In the case of BASIC, Microsoft's MBASIC has become the "standard" by which other BASICs were to be measured in speed and features. Standards for PASCAL were established by Jensen and Wirth's classic treatise, C by the work of Kernighan and Ritchie, COBOL by GSA, etc. A number of emulations of Bell Lab's UNIX operating system are now emerging in an attempt to establish standards by imitation.

Particularly interesting has been the evolution of "standards" in the realm of eight bit operating systems. CP/M-80® compatible operating systems have only recently developed the widespread title of *de facto* standards and then largely because of the large number of hardware configurations supported, and the many applications packages available in a wide range of media formats which "plug-into" this environment. And note that it has not been by virtue of these operating systems' sophistication but rather by lack of competition that they have achieved this popularity.

Now we find ourselves confronted by a unique situation. Just as IBM's communication, disk formats and other standards rapidly came into worldwide use as industry standards, SB-86™(i.e. Microsoft's MS-DOS™) is also showing significant signs of worldwide acceptance as a *de facto* standard. Particularly interesting was IBM's decision to establish this previously little known operating system as their standard. Certainly Microsoft's well earned reputation for quality as reflected by MBASIC, BASCOM, etc. were contributing factors in this decision. But perhaps most important to the industry as a whole is IBM's decision to license a non-exclusive operating system allowing other manufacturers to follow.

SB-86 has a number of highly important features which we shall discuss in later issues but it is clearly a technically superior operating system. Although IBM is licensed to distribute this O.S., Microsoft and Lifeboat are actively developing it and its market, to insure its broader popularity.

OEM's are showing renewed interest in the 8086 and many are preparing and/or revisiting plans to produce 8088/8086 environments as their next generation hardware. AMD's recent reversal might be the death blow to the Z8000. Authors are scrambling to gather the languages, tools and hardware required to move their applications into the SB-86 world and the press is hard at work preparing a variety of articles, new product announcements, review articles as well as new publications targeted for this market.

A crucial component of this rapid evolution is the proposal, adoption and implementation of extensions to SB-86 which brings us to perhaps one of the most exciting aspects of this new operating system.

SB-86 is an excellent candidate for a public standard for a sixteen bit operating system for desktop computers and should be give serious consideration along with other potential candidates.

Authors of other candidates are invited to place in the public domain functional specifications of their operating systems so that compatible and competitive products can also be considered. Any CP/M-86™ compatible range of operating systems might be at risk of claims of infringement of trade secrets and copyrights.

It is important to avoid a repetition of what happened with CP/M-80. As you recall, the world had quickly adopted the public domain version 1.0 of CP/M-80 as evidenced by the design of compatible operating systems such as those for Cromemco, Sanyo, Mostek, Infosoft, et al. Digital Research has however chosen to develop later versions in directions that have effectively destroyed this standardization, by declaring such changes as trade secrets or copyrighted. It is a sad commentary that users of CDOS, IMDOS, etc. are now confronted with a myriad of incompatibilities imposed by DR's attitude against shared standards.

SB-86, OASIS, CP/M-86 and others should all be considered candidates in whole or part in arriving at a future true standard in order to insure no recurrence of the confusion now existing in the CP/M-80 world.

Lifelines calls upon the readership and the industry to support a major movement to provide sufficient definition and scope to assure that industry standards emerge rapidly. We are anxious to support any and all efforts of this type and actively encourage you to make known your ideas and proposals. The benefits are many and obvious, not the least of which is the mutual advantage for all in establishing rigorous standards to assure applications of a rich and stable environment in this second generation of personal computers.

Therefore we encourage you to forward your questions, suggestions and proposals regarding sixteen bit operating systems such as SB-86 and their extensions to me at *Lifelines*.

This is a unique opportunity to influence the future of the industry and serves well the interest of all. We look forward to your comments.

Edward H. Currie

The Pipeline

by Carl Warren

Terminals offer many features

Terminals are key data entry peripheral devices to almost any system you can name. What to buy and why can be confusing, especially if you're unsure of what's available and of the different categories each terminal falls into.

A display terminal can be categorized under three headings: dumb, smart, and intelligent. The first category of terminals comprises those which provide a simple minded input device to a system much like a teletype. And in fact are frequently referred to as glass teletypes. The second category (smart) consists of those terminals that provide editing capabilities of line and character insert, plus the ability to delete. The third category of intelligent terminals includes those which give you the ability to program them; these are in fact stand-alone desktop computers.

Currently, there are approximately 150 different so-called dumb terminals available, ranging in price from $375 to $995, depending on screen size and functional capability. The smart terminal segment is similarly glutted with more than 120 possible choices. The intelligent terminal, though, is a different animal entirely, and only about 50 vendors offer products that fall into this category.

Although there is a great deal of interest in the low-end, under $1000, terminal, the intelligent terminal appears to be the best choice for certain applications, of which distributed processing leads the pack. But be aware, intelligent terminals don't come cheaply and represent a major investment – with prices ranging from as little as $2,000 to $8,000, again depending on capability.

In general the intelligent terminal offers such features as: full screen displays of 80x24 on a 15-in. screen, numeric keypads, and function keys, employment of dual processors (such as the Z-80 found in the Heath/Zenith Z-89, for example). In addition, 16-bit processors are just starting to find their way into terminals like Piiceon's PM-2010; it employs an Intel 8086.

Moreover, you can expect to find intelligent units that support up to 64K bytes of user memory and support a full range of system languages and a variety of operating systems. Not so surprising is that Digital Research's CP/M operating system appears to be the prime choice among manufacturers.

The problem that seems to occur with intelligent terminals, is whether the device is a terminal or a computer. Most vendors believe that the definition is broad enough to cover both fields. The Z-89, for example, is in most cases employed as a stand-alone desktop computer. But the unit serves as an excellent front end to larger mini or mainframe systems in a distributed environment.

Most observers expect the next generation of intelligent terminals to be almost minicomputer-like systems with 16-bit processors, Winchester disk drives, and either a floppy or tape serving as a backup device. In addition, current outlooks call for the employment of intelligent I/O using Intel's 8049, for example, to take the burden off the main processor.

Yes, color is making its way into the intelligent world also. Intelligent Systems Corp. possibly leads the pack with desktop color terminals, and Radio Shack is making a major dent in the market with the TRS-80 Color Computer. This latter system offers 16-bit capability in an 8-bit package. Radio Shack is carrying its offering forward by including along with the tiny machine: a disk drive, color plotter and high-resolution software packages with animation capability.

For those applications which don't require a lot of local intelligence, but need screen handling capability, you'd do well to consider taking a look at the so-called smart or editing terminals. These terminals are priced in the $850 to $995 price range and provide a number of capabilities for handling screen data in concert with an application package like MicroPro's WordStar, for example. Typically these terminals offer 80x24 screen formats, numeric keypads and function keys. What's missing are large local memories; a one or two page buffer is usually included, and disk functions. Some of the terminals that fall in this class and bear looking at include: TEC's models 530, and 560 priced at $995, Televideo's Model TVI-912 and TVI-920 carry prices tags of $845 and $995 sport 12-in. diagonal screens and with 1920 character display. Micro-term is in there with the ACT series; prices range from $675 to $995.

Virtually all the editing terminals provide similar characteristics of insert, delete, scroll, and highlighting. What confuses the issue, however, is the implementation of control and escape definitions. This latter problem has caused major implementation difficulties for companies like Organic Software, Sorcim and others who make generalized packages. According to Sorcim's Richard Frank, developing an install portion of a program takes as much time as the mainline coding, due to the different definitions.

The problem seems to center around the market pressures that cause terminal manufacturers to be forced into offering more bells and whistles. And one method of doing so is providing functions using escape and control codes that they define themselves rather than following accepted ANSI standards.

Interestingly, however, this may be changing, according to various industry observers. Many believe that because of the necessity of providing compatibility to a host of systems, and of operating both in the distributed and networking world, terminal manufacturers will have to begin providing industry compatibility in order to survive.

(continued next page)

Further muddying the terminal waters, are the introductions of small portable terminals. Products such as Novation's Infone, and Sony's terminal design. These terminals fall into the intelligent category since they offer storage, editing, and communications capability. But they aren't expected to impact the systems world because they are designed to be carried in attache cases for use anywhere.

Although these small portable devices are just now beginning to show up, expect to see by NCC next year a host of products from a variety of domestic and foreign manufacturers. It's expected that you'll be able to see a full range of terminals from Epson America including a flat screen display, and similar products from Sinclair, Sony and Hitachi. Reportedly, a number of 15-in. diagonal flat screen monitors have been shown in Japan during the past few months, and it is rumored that they are on their way here.

Communications are starting to play a big part in terminals. Lear Siegler is planning to lead the way by providing a less than $1000 add-on to their ADM-32A this month. However, they may be in for a surprise from a number of quarters including Televideo. It's expected, that by year's end a number of terminal manufacturers will be offering communications capability, ranging from 300 baud Bell 103 compatible units to Bell 212A devices, and providing both direct connect functionality and acoustic coupled options.

With the growing trend toward offering terminals with a host of "smarts" has come the necessity of including a variety of software features implemented in firmware. The IBM Personal Computer may be the first intelligent terminal system to be offering a number of user selectable character sets, as well as the necessary hooks for SNA, and X.25 communication protocols. These features already available in the system, but not reported, are expected to play an important role in future software enhancements coming from the giant computer maker.

Further enhancing both intelligent and smart terminals are facilities to provide 132 columns on the screen and as many as 66 lines. Currently, word processing systems that provide full page attributes call for rotating the screen to be vertical. Terminals from Datagraphix, Data General, Direct, Micro-term and Ontel, to name a few, offer the 132 columns and range in price from $1300 to $6000. The cost difference is dependant on how much intelligence is built into the machine.

Future enhancements are most likely going to be in the form of employing a greater number of and more power microprocessors in the terminal. In addition, expect color to begin supplanting monochrome displays even in the low-cost (under $1000) area. Already Intelligent Systems is preparing to offer a very smart color terminal for under $1500 by year's end.

Other enhancements to come will most likely include built-in transducers for connecting to Ethernet type networks, if not full node capability. Expect also to find a complement of serial interfaces from RS232C to RS449, and terminals that will offer transfer rates approaching 1Mbits/sec by mid '82.

Cost may be the most exciting news item of '82 since it appears terminal manufacturers are looking to reduce costs of even the most intelligent terminal to the under $1500 range.

Change of Address

Please notify us immediately if you move. Use the form below. In the section marked "Old Address", affix your <u>Lifelines</u> 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:		Old Address:	
NAME		NAME	
COMPANY		COMPANY	
STREET ADDRESS		STREET ADDRESS	
CITY	STATE	CITY	STATE
ZIP CODE		ZIP CODE	

8080 Assembly Language Tutorial—Part 3: Terminology and Architecture

by Ward Christensen

Editor's Note: We continue Ward Christensen's list of essential terminology, which began last month.

OBJECT PROGRAM

An object program is a PROGRAM which is in the form that the 8080 can use. This might be in paper tape format, or on cassette or floppy disk. A SOURCE PROGRAM must be run through the ASSEMBLER to produce an OBJECT PROGRAM which can then be executed on the 8080.

OCTAL

A numbering system in base 8. It is used because of the difficulty in working with BINARY. When 3 BITS are grouped together, they are described as an OCTAL digit, and can contain a value from 0 to 7. See the table listed under HEXADECIMAL.

OP CODE

Short for operation code, the name for an 8080 instruction. For example, the OP CODE 'INR' is used to increment the value in a REGISTER. (See also PSEUDO OP)

OPEN

Means to prepare a FILE to be processed. If you do not issue instructions to OPEN the file, then, for example CP/M doesn't know where it is on disk.

OPENing a file typically involves scanning a DIRECTORY to find the name, then bringing some information into memory, so that the location on the FLOPPY DISK to find the information.

OPERAND

The part of an ASSEMBLER INSTRUCTION which says which REGISTER, ADDRESS, or other data is to be used. For example, in the instruction to increment the A REGISTER 'INR A', the INR is the op code, and 'A' is the OPERAND, telling which REGISTER to increment.

OR

Used in programming, much as it is in common English: to mean "either" one condition, "or" another.

For example, "IF A = B **OR** C = D".

OR also refers to the combining of BITS, in an "either/or" fashion:

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

In an 8080, we are always ORing 8 BITS at a time. For example, to test if two one-BYTE registers are BOTH zero, we would "OR" them together. As we can see by the preceding diagram, the results will be zero only if ALL the bits were zero in both registers.

OVERFLOW

When a signed arithmetic value in a computer increases or decreases so much that it can no longer be held, it is said to OVERFLOW. For example, when considering a BYTE to contain a signed number, it can take on values only between −128 and +127. Thus, if we had 120 in a BYTE REGISTER, and added 10 to it, it would OVERFLOW. The results, taken as signed, would be -126.
(See also **CARRY.**)

PARALLEL

When transferring data, if each BIT of data is transferred on its "own wire", then the transfer is said to take place in a PARALLEL fashion.

The most typical example is a printer, which takes 7 BITs of data to print a character. The interface between the computer and the printer is PARALLEL if there is one wire for each BIT. (See also **SERIAL.**)

PARITY

There are certain times when it is appropriate to ensure that data is valid, i.e. that something didn't go wrong with it as the computer processed it.

This mostly applies to SERIALly sent data, where a "glitch" could change a BIT.

Parity means adding a BIT, to keep the sum of the bits in a byte, either EVEN or ODD.

For example, in ASCII, the value for an 'A' is 01000001. If we were dealing with EVEN parity, the byte would be OK as it is. If we were dealing with ODD parity, we would change the first bit to a 1, i.e. 11000001.

Let's say we were transferring data SERIALly, and received a byte: 01010000, and that we were operating with ODD parity. The fact that the byte has an EVEN number of bits on (2), says it has EVEN parity, and therefore is wrong. (We cannot, from this single byte, determine which BIT was wrong.)

The 8080 has several instructions for testing the PARITY of a byte.

PATCH

When programming in assembler, since your SOURCE PROGRAM has to be processed by first the editor, then the ASSEMBLER, if you made a minor mistake, you hate to go back through that process just to make a minor change.

To PATCH your program means to change the OBJECT PROGRAM to fix a BUG. Under CP/M, using DDT

(continued next page)

or SID (see DEBUGGING) makes this task quite trivial, **if** you are not **adding** instructions, since you DO have to go back through the editor and assembler to "move" things around in your program.

At times, you might want to allow for inserting instructions. You may do this by coding a NOP - a special instruction which is "**N**o **OP**eration". In the 8080, this is simply the value 00.

PERIPHERAL

A 'computerese' term meaning a device which attaches to a computer. This may refer to a cassette device, disk drive, teletype, keyboard, video display, etc.

POINTER

A term used to refer to a REGISTER PAIR which points to something in MEMORY. For example, if we are keying in data to a BUFFER, then we could talk of having REGISTER PAIR DE as a POINTER to the input buffer.

PORT

The 8080 has the ability to ADDRESS up to 256 input devices, and 256 output devices. The control and data for input/output devices goes through one of these 256 device ADDRESSes, called PORTs.

For example, you might have a keyboard connected to PORT 1 to read the data. You would likely also have some control BITs coming in on another PORT, typically 0. Thus you would input from PORT 0 to test if a key had been pressed, and if so, input from PORT 1 to read the data.

PROGRAM

A sequence of INSTRUCTIONS which tell the computer what to do. A PROGRAM may be written in ASSEMBLER, BASIC, FORTRAN, or other languages, or may be in machine language, which means that the programmer figured out the BIT patterns required for his PROGRAM, and wrote them in BINARY, OCTAL, or HEX.

PROGRAM COUNTER

A 16 BIT REGISTER on the 8080 chip which points to the INSTRUCTION to be executed. It is incremented 1, 2, or 3 times when an INSTRUCTION is executed, so that it points to the next INSTRUCTION. Certain INSTRUCTIONs can cause a totally new value to be loaded, thus changing the sequence of INSTRUCTION execution. Abbreviation: PC.

PROGRAMMING LANGUAGE

Refers to the language used to program a computer. This tutorial is covering Assembly language: where you know the internal instructions of the machine, and program in them using symbols or MNEMONICS, such as LXI instead of having to code "11 hex".

An incomplete list of other programming languages which I have known to run on the 8080 are: APL, BASIC, "C", COBOL, FORTH, FORTRAN, LISP, PASCAL, AND PL/I.

PROM

An abbreviation for Programmable Read Only Memory. Since the contents of RAM (see RAM) are lost when power to it is shut off, it is nice to have some PROGRAM which does not 'go away' under this condition.

The ROM, or PROM provides that, since its contents aren't lost. The PROM is able to be written, erased, and read. The erasing process usually entails placing the MEMORY under an ultraviolet light for a few minutes. Writing the PROM requires special HARDWARE, because higher than normal voltages and critical timings are usually required when writing PROMs. Newer PROMs are getting away from these requirements, which should mean that it will be easier and less expensive for hobbyists to 'burn' their own PROMs. ('Burn' is the more common expression used instead of 'writing', when referring to PROMs.) The most common PROMs which the hobbyist uses are the 2708, organized as 1K (1024) BYTES, and 2716, which is 2K or 2048 BYTES.

PSEUDO OP

In an assembler program, an OP CODE refers to an instruction of the microprocessor being programmed.

A PSEUDO OP is another type of OP CODE, which tells the ASSEMBLER something, not the "target" microprocessor.

An example: EQU, meaning equate. You might use EQU to set a value to be used later in the program, such as:

```
PORT EQU 0
```

Then later in the program, if you code:

```
IN PORT
```

This will be the same as if you had coded:

```
IN 0
```

except that, if someone else needs to modify your program, they don't have to find all the "IN" instructions, but can instead just change the one EQU statement.

Tutorial section (4) will completely discuss the PSEUDO OPs used by the CP/M 8080 assembler.

RAM

An abbreviation for Random Access Memory. This refers to the MEMORY in a computer which can be read, and written. The term may be misleading, as nearly all MEMORY is random access, whether it can be written or not.

READ
READ-ONLY
READ-WRITE

The process of getting data into a computer is usually called READing. It may be a BYTE from a TERMINAL, or a SECTOR from a FLOPPY DISK, etc.

READ-ONLY is just what it sounds like: something which may be READ, but not WRITTEN. For example, a PROM (see PROM) contains data or programs, which may be read, but (usually, or at least not accidentally) written.

A FLOPPY DISK may be considered READ-ONLY, either by HARD-

WARE such as a "notch" in the jacket which is detected by a switch, or through SOFTWARE which has a "BIT" set saying the disk may not be written upon.

READ-WRITE simply means something which may be BOTH read and written. Again it may apply to such things as MEMORY, or a FLOPPY DISK.

REGISTER

A part of the 8080 CHIP which can contain data. Each REGISTER is either 8 or 16 BITS long. There are 8 8-BIT REGISTERs, A, B, C, D, E, H, L, and PSW. (see PSW) You can store data in any of them. 3 REGISTER PAIRS exist: B and C, D and E, and H and L. These may be used to contain 16 BITs of information, such as is used to contain a MEMORY ADDRESS, or other data. The PROGRAMmer works directly with these REGISTERs, with MEMORY, and with the PERIPHERALS. There are other REGISTERS which are usually not used directly, the PROGRAM COUNTER and the STACK POINTER.

REGISTER PAIR

When it becomes necessary to deal with more than an 8 BIT data value, or to have a REGISTER containing a MEMORY ADDRESS, a REGISTER PAIR is used. Registers B and C, D and E, and H and L may be combined to make the "BC", "DE", and "HL" REGISTER PAIRs. They have their own instructions. For example, INR is used to increment the value in an 8 BIT REGISTER, while INX is used to increment the value in a 16 BIT REGISTER PAIR.

ROM

An abbreviation for Read Only Memory. This refers to MEMORY in the computer which contains PROGRAMs and/or data, which was placed in the MEMORY when it was manufactured. The contents of ROM cannot be changed except during the manufacturing process. Perhaps more common in hobbyist computers, is PROM (see PROM).

RS-232

A standard, literally "Recommended Standard number 232" of EIA, the Electrical Industry Association. It refers to the electrical and mechanical means by which SERIAL devices are interconnected. Most VIDEO displays, and printers, use this interface.

At a minimum, an RS-232 device needs to connect 3 wires: ground, transmitted data, and received data. Voltages range between −25 and +25 volts.

S-100

A name given to the standard physical and electrical usage of the 100 pins which interconnect circuit boards for certain microcomputers.

Originally designed for the first "popular" hobbyist microcomputer, the "ALTAIR", it was later called "S-100" by other manufacturers not wanting to "implicitly advertise" for ALTAIR by calling it the "ALTAIR BUS" (that's as much my opinion as it is a fact).

SECTOR

That portion of a FLOPPY DISK which is the least number of BYTES which may be transferred to or from MEMORY in one request. Typically, 128, 256, 512, or 1024 BYTES. A number of SECTORS make up a TRACK. See also FLOPPY DISK, and TRACK.

In CP/M, a SECTOR refers to 128 BYTES of DATA. Your assembler program READS and WRITES 128 bytes at a time. A portion of CP/M handles doing the actual I/O to the FLOPPY DISK, including handling larger-than-128-byte sectors.

SHIFT

The process of moving BITs of data left or right. For example, a BYTE of data representing the number 19, is shown as the BIT pattern:

```
0 0 0 1 0 0 1 1
```

If we shift it left, this means we insert a 0 at the right side, and throw away the leftmost bit:

```
0 0 1 0 0 1 1 0
```

Similarly, a right shift of the original number results in:

```
0 0 0 0 1 0 0 1
```

NOTE that shifting left multiplies the number by 2, and shifting right divides the number by 2. If in shifting left, a 1-bit is shifted off, then the result will not be 2 x the original, since data is lost. Similarly, if right shifting shifts off a 1-bit, the answer will be the INTEGER portion, i.e. won't contain the ".5": 19 shifted right is 9 since there is no way in a BYTE to represent the ".5" of "9.5".
(See also **CARRY**)

SERIAL

When data is transferred from one place to another (between computers, or between a computer and a video display or printer), if the data is transferred along a single wire, the interface is said to be "serial". MODEMs are serial devices, since it is not practical to connect to another computer in a remote location with 7 or 8 individual wires to transfer the BITs which make up a BYTE.

Serial transfer may occur at many speeds, which are measures by how many "bits" are sent down the wire in one second. The classic "Teletype" runs at 110 bits per second. Most MODEMs which hobbyists will encounter run at 300 bits per second, some going even to 600 or more. Locally attached (no modem required) printers and video displays typically run at 9,600 bits per second, serially.
(See also **PARALLEL.**)

SOURCE PROGRAM

In the context of this 8080 ASSEMBLER tutorial, a SOURCE PROGRAM will be a PROGRAM written in ASSEMBLER which is in 'human' readable form, i.e. contains LABELS, OP CODEs, OPERANDS and optionally, comments, such as:

```
MVI   A,8   ;SET LOOP COUNT
```

The SOURCE PROGRAM is run through the ASSEMBLER, which converts it to an OBJECT PROGRAM.
(See also **OBJECT PROGRAM.**)

(continued next page)

STACK

An area of computer MEMORY which is usually used to save the ADDRESS to return to a PROGRAM. When that PROGRAM calls a SUBROUTINE and wants control to return back to itself.

It is also used to save the contents of REGISTERs in a PROGRAM, such that they can be restored after being used.

The STACK is "LIFO", i.e. Last-In-First-Out, like a 'paper spindle', i.e. the last piece of paper you spindled is the first to be removed.

The 8080 chip has a 16 BIT register, called the STACK POINTER which keeps track of where the 'top' of the stack currently is. Note that the stack pointer must be initialized (pointed to the top of some free place of MEMORY) before it is used. This is because it moves **down** in use.

STACK POINTER

A 16 BIT REGISTER on the 8080 CHIP which points to an area of MEMORY for use in saving data and ADDRESSes.
(See also **STACK**.)

STATUS

This word is usually used in referring to whether or not a particular I/O device is ready. For example: "What PORT do you use for CONSOLE status?" This means, "Which PORT contains a BIT which may be tested to determine whether a character may be output to your console, or whether a key has been pressed and may be read?"

In CP/M for example, routines are supplied which allow testing the status of the keyboard, but testing the console output may not be allowed. Normally, you just have something to send to the console, so don't need to find out whether it is ready. A very few programs have HARD CODED I/O which tests to see if the console is ready for output.

STRUCTURED PROGRAMMING

Entire books have been devoted to this subject, so I can hardly do it justice in a few paragraphs. However, an overview:

STRUCTURED PROGRAMMING is a means, or "methodology" of programming, designed to maximize the readability and quality of programs, and thus minimize the programming errors.

It usually refers to TOP-DOWN programming, i.e. in which you make heavy use of SUBROUTINES, so that small parts of your program are very readable, rather than being just a long series of "in-line" instructions.

STRUCTURED PROGRAMMING usually implies programming in a "high level" language, such as "C", or COBOL, but MAY be done in assembler, too. For example, a SORT program might be written; instead of coding instruction after instruction "in-line", you might code:

```
    CALL SETUP
    CALL READ
    CALL SORT
    CALL WRITE
    JMP FINISH
```

The beauty of this structured programming, is the possibility of readily observing that **there are not bugs in that program as it stands**. Five lines of code are easily comprehended, and yes, they will properly sort, given that the SUBROUTINES do what they should.

Suppose then, that each SUBROUTINE were similarly broken up into pieces:

```
SETUP  CALL OPENINPUT
       CALL GETPARMS
       CALL OPENOUTPUT
       RET
```

Again, you can be reasonably certain there are **no bugs** in this routine, since it is so small and easy to grasp.

Of course, since a "CALL" doesn't "really" do any "work", you will **eventually** have to "write some other instructions" which actually **do the work**.

Theoretically, following this practice, you will write programs which are, yes, **longer**, and somewhat **slower**, but will be more **readable** and have **fewer bugs**.

SUBROUTINES

A series of INSTRUCTIONs which perform a specific task. It is also possible to simply code the necessary instructions exactly where they are needed, but frequently, this means a much longer, and less readable (less **structured**) program.

For example, suppose it takes 30 INSTRUCTIONs to READ a SECTOR from a FLOPPY DISK, test for errors, and return. Let's further say these 30 instructions occupy 70 BYTES. If this task is done in 10 places in your program, this will take a total of 700 BYTES. If however, the READ routine were made a SUBROUTINE, and simply CALLed 10 times, then the total memory would be 70 for the routine, plus 10 x 3 BYTES per CALL, or 100 bytes total.

In general, even without memory savings, using SUBROUTINES improves the readability of the programs and they are therefore encouraged.

SYNCHRONOUS

Like ASYNCHRONOUS, refers to **events**, but in this case, to events which either occur simultaneously, or which occur at regular intervals.

SYNCHRONOUS data communication occurs when the BYTES come at regular intervals, with no time between them. This is more efficient than ASYNCHRONOUS, but often requires more hardware, or more expensive hardware, to handle it.

TERMINAL

A device which attaches to a MICROCOMPUTER, and which has a keyboard and printer, or a keyboard and video display.

TRACK

That part of a FLOPPY DISK from which data may be READ or WRITTEN, while just spinning the disk.

A TRACK is made of SECTORS.
(See also **FLOPPY DISK**, and **SECTOR**.)

TTL

A family of integrated circuits, most frequently used in microcomputers.

Members of the family include such simple functions as "NOT", or such complex functions as counters, REGISTERS, etc. In general, the electronics of a MICROCOMPUTER are made of some "major" integrated circuits, such as the CPU, and MEMORY, connected together by TTL "support" circuitry, which helps "drive" the electrical signals down the BUS, and which "decode" for example, which address is to be read, etc.

VIDEO

Refers to a means of showing data from a computer, using a television-like display.

A VIDEO display may be character oriented, displaying many lines of data (8, 16, 24, to as much as 40). Each line is typically 40, 64, or 80 columns wide. The most common display is 24 lines of 80 characters, although 16 x 64 is very common as those were the first low cost ones readily available to microcomputer users.

WRITE

The process of getting data **out** of a microcomputer. You may be WRITING to a TERMINAL, FLOPPY DISK, etc.

XOR or EXCLUSIVE OR

The term "OR" usually means "either or". Since computers require instructions to be very explicit, when you mean "either but not both", you mean "EXCLUSIVE OR", frequently abbreviated XOR. Let's compare this with the OR:

-INPUT-		—OUTPUT—	
A	B	A OR B	A XOR B
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	0

Note OR and XOR are the same, except when BOTH the inputs are true.

This concludes the TERMS part of the tutorial. In the following section, I go into the architecture of the 8080, which primarily involves a discussion of a diagram showing the registers in the chip, and their usage.

THE ASSEMBLER

If you get stuck on any terms I use, they can be looked up in the "terminology" section of the tutorial.

There are several ways to get programs into a computer. The most primitive consists of using front panel switches, or a monitor program (typically in PROM). This involves knowing what the binary, octal, or hex equivalent of the necessary instructions is. Fortunately, we can use our computers to help us program.

For example, BASIC INTERPRETs the source program, producing the required results. This interpretation process makes the program less efficient, yet the high level of BASIC allows US to function more efficiently as programmers.

An ASSEMBLER takes our source program, and converts it into an OBJECT program, which we may run.

The assembler has two primary purposes:

1. To convert the OP CODES or MNEMONICS which we use, into the appropriate machine language, and
2. To free us from the burden of having to keep track of exact addresses of data and subroutines.

Without an assembler, we would have to hand-translate the op codes into binary, and would have to calculate where data and routines were. If we change our program by adding or deleting instructions, then we have to go back and hand re-calculate all address references.

An assembler program consists of one-line statements. I emphasize *one-line*, because some programmers on large machines are used to the concept of "continued" lines, i.e. in which a single line may be continued for several lines. This is not true in a microcomputer assembler.

The statements may be either COMMENTS, or SOURCE LINES. Referencing the CP/M assembler, and most others, a comment line begins with a ";". It typically need not be in the first column, but usually is.

Comments may also be placed on the same line as an instruction, again using ";" to signify the end of the instruction, and the start of the comment.

A line may begin with a LABEL, which is used for two purposes:

1. To specify a name under which a particular value is to be stored, BY THE ASSEMBLER.
2. To provide a symbolic name by which a routine may be subsequently referenced.
3. To provide a symbolic name by which data may be subsequently referenced.

Following the label is the OP CODE, i.e. the instruction to be executed. If there is no label, spaces or a TAB, are used to skip to the op code.

Most op codes require one or more operands. Again, spaces, or more commonly, a tab is used to separate the op code from the operands.

Finally, whether there were operands or not, the optional comments appear. Again, a tab is used to set the comments off from the operands. If the op code required no operands, then two tabs are usually used, so the comments line up. The comments begin with the character ";".

Since this tutorial is being printed in fairly narrow columns, I will be using just a few spaces, where you will be using tab characters. Let's look at a few lines of a program:

(continued next page)

```
;
;DEFINE ADDRESS OF CHARACTER-OUT
;     ROUTINE IN MY PROM
;
TYPE  EQU  0F0C2H ;PROM ADDR
;
;
;PRINT THE MESSAGE
;
PRSUB LXI  H,MSG ;POINT TO MSG,
      CALL PRTHL ;PRINT IT
      RET        ;RETURN
;
MSG   DB   'This is a msg',0
;
;PRINT THE MESSAGE POINTED TO
;      BY THE HL REGISTERS
;
PRTHL MOV  A,M   ;GET CHAR
      CALL TYPE  ;TYPE IT
      INX  H     ;POINT TO NEXT
      MOV  A,M   ;GET IT AGAIN
      ORA  A     ;IS IT 0?
      JNZ  PRTHL ;NO, LOOP
      RET        ;OTHERWISE, END
```

You can see that if you had tried to assemble this program by hand, which IS possible, then decided to make the message larger, you would have to remember to change the address of PRINT which is in the CALL PRTHL instruction.

You would probably make a table of the names you used, i.e. the address of each. This would be the SYMBOL TABLE. When you changed your program, you would change the symbol table, so you could properly assemble the new address into the CALL PRTHL.

The CP/M 8080 assembler performs this function by reading your program twice. (Thus the term "Two Pass Assembler" which is frequently applied to it). The first time, it produces the symbol table. The second time, using the symbol table, it generates the object program.

In more detail: on the first pass, it reads each line. If it is a comment, it ignores it. If it is an EQU (EQUATE), it determines the value, and stores it into the symbol table. If it is a LABEL on an instruction or data, it puts that label into the symbol table, with the current "program counter" address.

If the line contains an instruction, it adds the length of the instruction to the program counter. If the line contains data, adds the length of the data to the program counter.

In the second pass, when it encounters an instruction like LXI H,MSG, it goes to the symbol table, looks up the address of MSG, and generates that address into the LXI instruction OBJECT CODE.

I don't plan to cover what code gets generated by each instruction, but will give this one example: LXI H,(something) generates a 21 HEX followed by the address. Being an 8080, the address is stored in memory, LOW BYTE FIRST, then high byte.

Thus, if MSG was at 1234 hex, the LXI H,MSG would generate:

21 34 12

Note that this is a 3-byte instruction, so the assembler, in its first pass, would have added 3 to the program counter when the LXI was seen.

THE CP/M ASSEMBLER

Future sections of the tutorial will teach you the 8080 operation codes. In the remainder of this section, I'll cover the PSEUDO OPs which the CP/M assembler covers.

Briefly, they are:

ORG	Set origin for code generation
EQU	EQUate some symbol to a particular value
SET	Similar to EQU, except that more than one value may be assigned during one assembly
IF	Conditionally generate the following instructions
ENDIF	End of block of conditionally generated instructions
DB	Define byte constant
DS	Define uninitialized storage area
DW	Define word constant

These are all adequately explained in the standard CP/M manuals. Let me add just a few comments:

When using IF with the CP/M assembler ASM, you may not "nest" them, i.e:

```
IF        XXX
•
•
IF        YYY
•
•
ENDIF           ;YYY
ENDIF           ;XXX
```

This *does* work with MAC, the CP/M Macro assembler, but not with ASM. The reason it doesn't work with ASM is that a single flag is used to hold the value of the last IF statement. Thus the *first* ENDIF encountered, makes ASM think it is no longer processing within an IF.

Another tip: DB's BYTE values may not be negative, because ASM evaluates numeric operands of the DB to a 16 bit value, then rejects it with an error message if any of the bits in the high byte are on. To circumvent this, code:

```
MVI      A,-5 AND 0FFH
```

instead of

```
MVI      A,-5
```

which would cause an error.

The "AND 0FFH" "ANDs" the 16 bit value for -5, namely in bits:

1111 1111 1111 1011

with the value 0FFH, which is:

0000 0000 1111 1111

producing:

0000 0000 1111 1011

which is acceptable as a DB operand.

Do you need some back issues or would you like to present a friend with one of your favorites? See page 37.

Custom I/O Routines For PIP

by Michael J. Karas

Any number of special applications exist for custom hardware I/O routines to be used under the CP/M operating system for inputting data, or for sending output data to physical devices not supported by the user's CP/M Custom Basic I/O System (BIOS). The possibilities range from simple, immediate, one-shot special requirements, to those cases where I/O must be implemented on a system with no source code for the BIOS available. Digital Research has provided an easy-to-implement custom I/O module capability within the PIP.COM utility. With this capability input to a special device may be referenced with the INP: physical device name. Likewise, output may be sent to a custom physical device with reference to the OUT: physical device name.

A typical application for such a procedure came to my attention recently in a discussion with a friend over a beer. The problem, simply stated, was how to get the default standard public domain MODEM program up and running on a new OSBORNE computer. Note that the 5 1/4 inch diskette medium is known not to be compatible with other systems. And another machine with the software up and running was not available. The obvious "not so desirable" solution was to type in the MODEM.ASM source code. That could take a long time. I suggested hooking the RS-232 port of the OSBORNE to an RS-232 port on a machine that had the source for MODEM available. Assuming that the hardware connection problem is not an insurmountable hurdle, the "immediate" mode patching of PIP.COM on the OSBORNE with DDT.COM could make the loading of MODEM into the OSBORNE with a simple command like:

A>PIP MODEM.ASM=INP:[B]<cr>

The [B] being used to buffer the source in memory and alleviate the need to perform hardware handshaking between the computers.

I do not know whether my friend implemented the idea described above, but after noting his amazement that the INP: and OUT: capability existed within PIP.COM, I decided to present an example implementation of the capability. The application here interfaces the 8251A USART on a Monolithic Systems MSC-8009A multibus Z-80 board with the INP: and OUT: references. The example is presented purely as an example and should not limit your imagination as to how you may get a special interface working on your machine.

A quick look at PIP.COM with DDT as shown in Figure 1 illustrates the patch area and INP: and OUT: interface scheme. This example shows the PIP patching procedure by entering the source code for the custom I/O drivers as an assembly language source file. This file is then assembled into a ".HEX" file. The program listing in Figure 2 shows the MCS 8009A serial port patching example. Note that the equate "PIPENTRY" in the file must be set to the address presently at locations 0101H and 0102H in your version of PIP. The example shown is fully compatible with the PIP programs supplied with both CP/M Versions 1.4 and 2.2.

The procedure for making a patched PIP.COM would be as shown in Figure 3. In the example, lower case characters are generally those typed by the operator and <cr> indicates depression of the carriage return.

The patched PIP is now ready to use. For initial debugging, the .PRN file listing of the patch assembly may be used in conjunction with DDT to test the new drivers.

```
A>ddt pip.com<cr>
DDT VERS 2.2
NEXT  PC
1E00 0100
-d100,23F<cr>
0100 C3 CE 04 C9 00 00 C9 00 00 1A 00 00 00 00 00 00 ................
0110 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0120 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)

             .... AND SO ON AND SO ON ...
            (Note that all of space from 010AH to 01FFH
             is available for patch area.)

01D0 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
01E0 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
01F0 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0200 20 20 20 43 4F 50 59 52 49 47 48 54 20 28 43 29    COPYRIGHT (C)
0210 20 31 39 37 39 2C 20 44 49 47 49 54 41 4C 20 52  1979, DIGITAL R
0220 45 53 45 41 52 43 48 2C 20 20 50 49 50 20 56 45 ESEARCH,  PIP VE
0230 52 53 20 31 2E 35 03 01 06 01 00 24 24 24 20 20 RS 1.5.....$$$
-l100<cr>
  0100  JMP  04CE     <==== Make note of this address and
  0103  RET    <==\         set equate "pipentry" in the
```

(continued next page)

```
 0104  NOP           \              patch file to this value.
 0105  NOP            \ ========= Pip call point for INP:
 0106  RET   <================= Pip call point for OUT:
 0107  NOP
 0108  NOP
 0109  LDAX D  <============== Pip expects INP: characters
 010A  NOP                      here with parity stripped.
 010B  NOP
 010C  NOP
;-----------------------------------------------------------------------
; DIRECT PIP I/O PATCH PROGRAM FOR PIP 1.4 AND PIP 2.2 INP: AND OUT:
;-----------------------------------------------------------------------
;
;      THIS PATCH FILE IS USED TO OVERLAY THE FIRST PORTION OF THE
;      DIGITAL RESEARCH "PIP.COM" FILE TO PERMIT THE BUILT-IN PIP
;      PHYSICAL DEVICE REFERENCE NAMES "INP:" AND "OUT:" TO BE
;      USED. THE ENTRY INFORMATION IS AS FOLLOWS:
;
;      A) INITIAL EXECUTION ENTRY OF PIP FROM ADDRESS 0100H PASSES
;         CONTROL INITIALLY TO AN INITIALIZATION ROUTINE TO INITIALIZE
;         THE CUSTOM I/O DEVICE(S). COMPLETION OF INITIALIZATION PUTS
;         CONTROL TO THE NORMAL PIP ENTRY POINT.
;
;      B) IF PIP.COM IS CALLED WITH A COMMAND LINE LIKE:
;
;                A>PIP FILENAME.TYP=INP:<cr>
;
;        PIP WILL EXPECT INPUT FROM AN I/O ROUTINE PATCHED INTO
;        PIP.COM BY CALLING I/O ROUTINE ENTRY POINT AT ADDRESS 0103H.
;        THE INPUT CHARACTER IS PASSED BACK TO PIP IN LOCATION 0109H.
;        PARITY MUST BE STRIPPED FROM INPUT CHARACTERS.
;
;      C) IF PIP.COM IS CALLED WITH A COMMAND LINE LIKE:
;
;                A>PIP OUT:=FILENAME.TYP<cr>
;
;        PIP WILL SEND OUTPUT TO ROUTINE PATCHED INTO PIP.COM BY
;        BY CALLING I/O ROUTINE ENTRY AT ADDRESS 0106H. THE OUTPUT
;        CHARACTER IS PASSED FROM PIP IN THE (C) REGISTER.
;
;
;      THIS DEMONSTRATION VERSION OF CUSTOM PIP PATCHING ASSUMES THAT
;      IT IS DESIRED TO PATCH PIP TO UTILIZE THE SECOND SERIAL I/O
;      PORT OF A MONOLITHIC SYSTEMS INC MSC-8009A Z-80 CPU CARD. THIS
;      PORT IS AN 8251A USART WITH BAUD RATE CONTROLLED BY ONE OF THE
;      COUNTERS IN AN 8253 CHIP. THE OPERATION HERE DEMONSTRATES USE
;      OF THE PORT FOR BOTH INPUT AND OUTPUT FUNCTIONS. IT IS INITIAL-
;      IZED TO 300 BAUD, 8 BITS NO PARITY, AND ONE STOP BIT.
;
;-----------------------------------------------------------------------
;
;
;MSC 8009A SECOND SERIAL CUSTOM PORT CONFIGURATION
;
;   RS 232 PORT EQUATES FOR MSC 8009 Z-80 BOARD
;
;
CCTRL    EQU       0CFH              ;CUSTOM 8251A CONTROL PORT
CSTAT    EQU       0CFH              ;CUSTOM 8251A STATUS PORT
CDATA    EQU       0CEH              ;CUSTOM 8251A DATA PORT
;
```

```
TCC       EQU      0DFH                ;CUSTOM 8253 TIMER CONTROL PORT
TCR       EQU      0DDH                ;CUSTOM 8253 TIMER REGISTER PORT
TCCW      EQU      076H                ;CUSTOM 8253 TIMER CONTROL WORD
;
SRRDY     EQU      002H                ;8251A RECEIVER CHARACTER READY MASK
SRVAL     EQU      002H                ;8251A RECEIVER READY VALUE
STRDY     EQU      001H                ;8251A TRANSMITTER EMPTY READY MASK
STVAL     EQU      001H                ;8251A TRANSMITTER READY VALUE
;
INITC1    EQU      040H                ;8251A INITIALIZATION ...
INITC2    EQU      04EH                ;...
INITC3    EQU      037H                ;... ALL THREE OF 'EM
;
DCBR      EQU      416                 ;DEFAULT FOR CUSTOM PORT AT 300 BAUD
;
;SET PIP JUMP ADDRESS FROM YOUR COPY OF PIP.COM AS VIEWED BY DDT.COM
;AT ADDRESS 0100H. THE FOLLOWING EQUATE IS SET TO THE JUMP ADDRESS
;FROM LOCATION 0102H/0103H.
;
PIPENTRY EQU       00000H              ;SET BEFORE ASSEMBLY
;
;
;ESTABLISH ENTRY POINTS FOR OVERLAY OF PIP.COM PATCH FILE
;
          ORG      0100H               ;BEGINNING OF PIP.COM
          JMP      INITIALIZE          ;GO HANDLE INITIALIZATION OF SIO
;
          JMP      CI                  ;CUSTOM INPUT PORT ENTRY LOCATION
          JMP      CO                  ;CUSTOM OUTPUT PORT ENTRY LOCATION
;
RETCHAR:
          DB       01AH                ;PIP INPUT RETURN CHARACTER LOCATION
;
;
;HERE TO SETUP OUR SERIAL PORTS AND THEN PASS CONTROL DOWN INTO PIP.COM
;
INITIALIZE:
          CALL     SINIT               ;SUBROUTINE TO INITIALIZE USART
                                       ;..AND TIMER.
          JMP      PIPENTRY            ;OFF TO PIP
;
;
;CUSTOM SERIAL I/O INITIALIZATION ROUTINE
;
SINIT:
;
;SETUP TIMER COUNTER CHIP FOR BAUD RATE CLOCKS
;
          LXI      H,DCBR              ;GET CUSTOM PORT BAUD RATE CODE
          MVI      A,TCCW              ;GET CUSTOM PORT TIMER MODE WORD
          OUT      TCC                 ;SEND IT TO TIMER
          MOV      A,L                 ;GET LSB'S OF BAUD RATE CONSTANT
          OUT      TCR                 ;SEND
          MOV      A,H                 ;HIGH BYTE
          OUT      TCR                 ;SEND THAT ALSO
```

(continued next page)

```
;
;
;INITIALIZE THE 8251A WITH TRIED AND TRUE METHOD
;
         MVI      B,020                ;LOOP COUNT
         XRA      A                    ;NULL
INITLP:
         OUT      CCTRL                ;RESET 8251A'S ...
         DCR      B
         JNZ      INITLP               ; ... 'TILL THEY'RE GOOD AND DEAD
;
         MVI      A,INITC1             ;SEND THREE ...
         OUT      CCTRL                ; ... INITIALIZATION CHARACTERS ...
         MVI      A,INITC2             ; ... TO 8251A'S
         OUT      CCTRL
         MVI      A,INITC3
         OUT      CCTRL
         IN       CDATA                ;PURGE UART GARBAGE
         IN       CDATA
         RET                           ;FINALLY DONE WITH ALL THAT
;
;
;CUSTOM PORT INPUT ROUTINE
;        GETS CHAR TO PIP RETURN LOCATION AT (0109H)
;
CI:
         IN       CSTAT                ;GET READY STATUS
         ANI      SRRDY                ;MASK RECEIVER READY
         CPI      SRVAL                ;COMPARE WITH READY VALUE
         JNZ      CI                   ;REPEAT TILL INPUT READY
         IN       CDATA                ;GET INPUT IF READY
         ANI      07FH                 ;STRIP PARITY
         STA      RETCHAR              ;PUT INTO PIP RETURN LOCATION
         RET
;
;
;CUSTOM PORT OUTPUT ROUTINE
;        SENDS CHAR IN (C) REGISTER
;
CO:
         IN       CSTAT                ;GET READY STATUS
         ANI      STRDY                ;MASK XMITER READY
         CPI      STVAL                ;COMPARE WITH READY VALUE
         JNZ      CO                   ;REPEAT TILL XMITER EMPTY
         MOV      A,C
         OUT      CDATA                ;PUT DATA OUT NOW THAT READY
         RET
;
         END
;
;
;+++...END OF FILE
```

```
B>ddt pip.com<cr>
DDT VERS 2.2
NEXT  PC
1E00 0100
-ipippat.hex<cr>
-r<cr>
NEXT  PC
1E00 0000                  <=== Note ending PIP address so to SAVE
-d100,17f<cr>                   01EH - 1 or 30 pages after patching.
                           <=== Dump to see installed patch.

0100 C3 0A 01 C3 37 01 C3 48 01 1A CD 10 01 C3 00 00 ....7..H........
0110 21 A0 01 3E 76 D3 DF 7D D3 DD 7C D3 DD 06 14 AF !..>v..}........
0120 D3 CF 05 C2 20 01 3E 40 D3 CF 3E 4E D3 CF 3E 37 .... .>@..>N..>7
0130 D3 CF DB CE DB CE C9 DB CF E6 02 FE 02 C2 37 01 .............7.
0140 DB CE E6 7F 32 09 01 C9 DB CF E6 01 FE 01 C2 48 ....2..........H
0150 01 79 D3 CE C9 2F 4F 55 54 3A 53 50 41 43 45 29 .y../OUT:SPACE)
0160 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0170 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)

-^c                        <=== Exit DDT to system
B>save 30 pippat.com<cr>  <== save off patched PIP.COM
```

KIB-BITZ

Tips & Techniques

by Ward Christensen

If you are "comfortable" with the terms:

* CP/M disk allocation (1K for single density, 2K or more for double)
* XSUB.COM, SUBMIT.COM and $$$.SUB
* That SUBMIT writes one command per sector, and processes them "backwards".
* SAVE n filename.type

then skip to "IMPLEMENTATION".

CP/M Disk Allocation

CP/M allocates the disk in "chunks", usually called BLOCKS or GROUPS. In single density, these are 1K (8 sectors) long, and in double density, "usually" 2K (16 sectors) long.

SUBMIT

CP/M SUBMIT.COM takes a file of commands, and writes them to disk, to a file called $$$.SUB, in REVERSE order. This cleverness (score another one for Dr. Gary Kildall) is done in order to efficiently process the file. Here's how:

1. SUBMIT fetches the "next" command, by OPENing $$$.SUB, taking the file size byte (number of sectors) from the FCB and uses that as the sector to read.
2. It reads the sector into the command buffer, just as it would appear if you had typed it. (Thus it had been stored out on the disk, one command per sector, in the form "nn command 00" where nn is the 1-byte length, in binary, and "0" is a final binary 00.
3. It then DECREMENTS the file size by 1, (so next time it reads the "last sector in the file", it will get the PREVIOUS sector).
4. It CLOSES $$$.SUB, writing its directory entry back.

Of significance to this programming tip, is that CP/M accesses the disk DIRECTORY, then the $$$.SUB file, then the DIRECTORY, for EVERY command line processed.

XSUB, (Not available in CP/M 1.4), is used to allow entry of sub-command lines to PIP, DDT, SID, or other programs that call the BDOS "read line" function. It "hangs itself" under CCP, and reads commands for these programs, which would usually be typed at the console.

For example, if you want to PIP 8 files from A: to B:, but they are not able to be specified by any "*" or "?" type name (i.e. not *.asm), then you might want to create a list of the names with the CP/M editor (or WordMaster, or my FMAP program which writes directory names to a disk file, etc), so it looks like:

```
XSUB
PIP
B:=A:MAST.CAT
B:=A:ORDER
B:=A:OUTLINE
B:=A:PREPROC.BAS
B:=A:PREPROC.PRE
B:=A:REPLY.LTR
B:=A:SPELL.ASC
B:=A:SPELL.BAS
```

When you then SUBMIT this file, XSUB hooks itself in, and returns. CCP, the console command processor, then reads the next line, and loads PIP.

XSUB then takes over, and reads the following lines one at a time, giving them to PIP. This is far superior in performance to an ordinary SUB file looking like:

```
PIP B:=A:MAST.CAT
PIP B:=A:ORDER
PIP B:=A:OUTLINE
PIP B:=A:PREPROC.BAS
PIP B:=A:PREPROC.PRE
PIP B:=A:REPLY.LTR
PIP B:=A:SPELL.ASC
PIP B:=A:SPELL.BAS
```

because with XSUB, PIP doesn't have to be loaded again for each command line.

SAVE 1 X

My Submit-speeder-upper idea makes use of the SAVE command, to "reserve" a place on disk. For example, if you SAVE 1 X on a blank disk, then the first 1K of the disk (or 2K or more in double density) will have "X" as its name, and nothing else will go there. This not only reserves the first 1 or 2K on disk, but also reserves an entry in the first directory block.

If later, when the disk is fairly full, you want to make an "opening" near the front of the disk, (see why, later), just temporarily ERA X. If you should want to reserve more than the minimum 1 or 2K, then remember SAVE takes as an operand, the number of PAGES (256 bytes each) to save. For example, in my double density system, if I save from 1 to 8, it reserves 2K, or 16-128 byte sectors. If I save 9, it "reserves" 4K, or 32 sectors.

IMPLEMENTATION

The "overhead" time necessary to process a long SUB file under CP/M is very dependent upon WHERE on the disk the temporary file ($$$.SUB) created by the SUBMIT command is stored.

As a "worst case" example, if you had a "nearly full" disk, and the temporary $$$.SUB file had to be written to track 76, then processing each command would require:

1. A seek to the directory (track 2) to open the file;
2. A seek to track 76 to read the command;
3. A seek back to the directory to close the $$$.SUB file, making it one shorter.

The "obvious" solution is to "force" the placement of the $$$.SUB file "closer" to the directory. How might this be done?

1. When creating a system disk, after formatting it, do "SAVE 1 X".
2. Keep the "place holder" file "X" on your disk at all times.
3. When you are about to issue the SUBMIT command, type: "ERA X". This will "open up" a place for the $$$.SUB file to "sit".

On a single density disk, this means that up to 8 SUBMIT commands will be processed, QUICKLY, since it will be able to go between the directory and the $$$.SUB file, WITHOUT A SEEK. You may use your "own creativity" in setting up and using the "X" file.

For example, if you frequently have VERY LARGE SUBMIT files, then 1K, make your save higher. You could also save multiple files, such as "X" and "X1". Then, when you need 8 or less entries for a submit, ERA X. For 9 to 16, ERA X then ERA X1. You wouldn't want to just have a "very big" "X" file, as frequently you are placing things ONTO the disk which contains "X", and if all of "X" were not filled with the $$$.SUB file, you would be unable to place "X" back since some of the files written to the disk might have "filled in" the space normally reserved for "X".

I take a more simple approach: under double density, I SAVE 9 X so that I have 16 sectors available for $$$.SUB. This is usually sufficient.

Another way of saving time is to consider how to "minimize" the number of lines entered. For example, in testing CBBS® , the Computerized Bulletin Board System, I want to have two versions of the .COM file, namely an ON-LINE version, and a TEST version, for testing at home without a modem.

Formerly, I used a TEST EQU TRUE flag in the assembly, but that took a complete assembly, only to change perhaps a dozen instructions and ten flags. So, I created a SUB file for use with SID, that "patched" the live version, to make it the test version. For example, SID allows:

```
s.welcom
0
.
```

to mean "store a 0 into the byte at label WELCOM". This takes 3 lines, and thus, under XSUB, 6 directory accesses, and 3 accesses of $$$.SUB. I then realized the single line (true for SID only):

```
f.welcom,.welcom,0
```

would do the same thing.

Similarly, one day I was using XSUB to PIP certain files from one disk to another, so I made a list of all of them. I then realized I was PIPping ALL the .ASM files, all the .DOC files, and an assortment of others. So I changed my SUB file to contain all specific names which I couldn't specify with "*", such as:

```
B:=C:MAILLIST.INT
```

but then deleted all the individual .ASM names, replacing them with:

```
B:=C:*.ASM
```

and similarly with .DOC files, have just:

```
B:=C:*.DOC
```

By employing these techniques:

1. Reserving a "slot" near the directory for the $$$.SUB file, and
2. efficiently using the SUB file entries,

I am able to significantly speed up many of the functions I call upon my system to perform.

SID/DDT with .COM files which return to CCP

Skip this paragraph if you know what CCP is. CCP is the CP/M Console Command Processor, which handles the TYPE, DIR, ERA, SAVE, REN, and USER built-in commands, and loads and executes other programs such as ASM, MBASIC, etc.

There are 2 principle ways in which CP/M .COM files "end" their execution:

1. They may "reboot" by JMPing to 0, or by issuing BDOS call 0; or
2. They may simply "RETurn" to CCP (usually having established their own STACK, then restoring the CCP stack before returning).

Typical of case (1) are: ED, ASM, and MBASIC, which want to make use of all available memory, thus overlaying CCP to use memory up to the bottom of BDOS. Typical of case (2) are: STAT, and most user programs. Special consideration must be given when testing case (2) programs under DDT or SID. Specifically, since they "RET" (return to CCP) you must supply them something to return to.

When you run your program under normal CP/M operation, CCP "CALLS" your program i.e. you may RET to CCP.

However, when you test your program with DDT or SID, the stack pointer is set at 100H, so that it uses 0FFH downward for a stack. Also, no return address is supplied if you execute your program via "G100". Don't be

(continued next page)

tempted to "read between the lines" in SID, and "C100" i.e. call the program at 100:. This call facility is used for SID utilities, and you *may* use it, but no breakpoints or pass points will be set.

Here's how to get around that: Set up some portion of high memory as a stack. For example, I have a 2K block at 0F800H which I frequently load special programs into. I type in the following at 0F000H:

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

After preparing to execute the program at 100H (such as by placing names in the FCBs with the DDT/SID "I" command), I type:

```
    GF000
```

which executes my patch, loading the stack pointer, and calling the program at 100H.

When the program has finished execution, it returns, hits the "RST 7", which returns control to DDT or SID.

As an aside, if I am debugging a program with SID which returns to 0 when it is done, I do:

```
    p0
```

to set a 'pass point' at 0. Then, when debugging the program, if it should end, I can 'regain control' via the pass point at 0.

Tip Contest

Did you know that those tips and techniques you've created may be valuable to others and worth $$ to you? Our tip-of-the-month winner receives a $50 prize. So send your tips and techniques, in machine-readable form, to Lifelines Tip Contest, 1651 Third Avenue, New York, N.Y. 10028. We're looking forward to hearing from you.

```
;Name:              MOVER.ASM
;Author:            Ward Christensen
;Written:           05/25/81
;Function:          Data movement subroutine which detects
;                   if it is running on a Z-80, and does
;                   an LDIR if so. Otherwise, 8080
;                   instructions are used
;Processor:         ASM
;Dependencies:      None. 8080/Z-80/8085
;Revs (last first):
;                   (NONE)
;Usage:
;                   On entry to the routine:
;                            HL = Source field address
;                            DE = Destination field address
;                            BC = length to be moved.
;
;Logic:             The 8080 sets the parity flag on any
;                   arithmetic or logical instruction.
;                   The Z-80 sets it only on logical
;                   instructions. Arithmetic instructions
;                   use this bit as an overflow bit, so
;                   something like MVI A,2, INR A, will reset
;                   it as an overflow flag on a Z-80, but
;                   set it as even parity, on an 8080.
;
;
;I INVENTED THIS ROUTINE WHILE SWITCHING BETWEEN AN
;8080 AND A Z-80 BOARD, WHILE NOT WANTING TO HAVE TO
;REASSEMBLE THE DATA MOVES DONE BY MY BIOS VIO SCROLL
;ROUTINE AND THE DOUBLE DENSITY DEBLOCKING ROUTINE.
;
;N-O-T-E THE 3 INSTRUCTIONS MAY BE USED WITHOUT THE MOVE
;         IN OTHER PROGRAMS:
;
;        MVI     A,2
;        INR     A
;        JPE     XXXXX    ;JMP IF 8080      (E)IGHTY-EIGHTY
; or                                        AS A MEMORY-AID?
;        MVI     A,2
;        INR     A
;        JPO     XXXXX    ;JMP IF Z-80
;                ===============
;
MOVER    MVI     A,2      ;A = 2
         INR     A        ;                 SET PARITY ONLY IF 8080
         JPE     MOV8080  ;GO TO THE 8080 MOVE
         DB      0EDH,0B0H ;THIS IS THE Z-80 MOVE, LDIR
         RET
;
MOV8080  MOV     A,M
         INX     H
         STAX    D
         INX     D
         DCX     B
         MOV     A,B
         ORA     C
         JNZ     MOV8080
         RET
;
```

After the Game

by Stephen Walton

Editor's Note: Stephen Walton's story represents our first foray into the world of fiction. We're anxious to hear what you think of this story, and what your feelings are about an occasional fiction feature.

It amused him to reign as Thorvald II. The mess he had inherited did not amuse him, but it could be managed. He spent heavily on economic development, made sure of adequate patrol, and cut taxes. Three star systems rebelled anyway, but the Navy brought them back into line without loss of ships.

In the second cycle, he continued his established policies — spending to cook the economy, fleets sufficient for patrol but not intimidating in their size, low taxes — and the Empire responded well. The Gross Imperial Product went up substantially, while the Index of Unrest compiled by his Secret Service went below the danger threshold by a comfortable margin. He was popular, and there was no more rebellion.

Within a few cycles, prosperity was a fact. Now he could turn his attention outward, and try to make the Empire grow. He held his exploration spending to a small, fixed proportion of the total budget, knowing it could have an unfavorable effect on the political climate. He lost a few popularity points, but the Index of Unrest remained in the safe range.

His Empire grew, some cycles by just one or two usable star systems, some cycles by 18 or 24 or more. The Navy was gradually enlarged to patrol what he ruled, so that piracy would not take root, and the economy performed well.

Then one of his exploring groups encountered the insanely destructive Licorice Empire, and was wiped out. He committed the Navy. The Licorice, who had the advantage of numbers, destroyed a dozen of his star systems and nearly a hundred of his battle cruisers. But the Navy kept them from overrunning the Empire and bought him time. He built a thousand new ships, and with them he crushed his enemy.

"Hey Thorvald, nice going," said Peter.

"It's gotten almost too easy," said Robin Harris, who had told the computer his name was Thorvald II.

"Well, you've got the hang of it by now. Don't forget to decommission the ships you don't need for patrol, or your U factor will skyrocket."

"Right." Harris made the appropriate keyboard entry.

"How did you like the bad guys?" Peter was grinning.

Harris looked over his shoulder at his host. "I knew you had to have something up your sleeve, with the machine asking me for 'someone or something you don't like.' "

Peter nodded. "Of course. You just didn't do well enough the other times to get a chance to meet them."

"Don't rub it in."

"Do you really like the game?" Peter was looking at Harris anxiously; he was twenty-eight years old and had had his own computer for three months now. Harris had finally set aside an evening for seeing what the machine could do.

"Yes, I like it," Harris said. "It does get you involved. By the way, what time is it?"

Peter glanced at his digital watch. "Ten twenty-six."

"Good God, I've got to catch a train." Harris stood up and stretched his arms over his head. "I had no idea it was getting so late."

Peter was grinning again. "That's what these machines do best — they *eat time.*"

"Uh-huh," said Harris, taking his coat from the bed in the middle of the cluttered studio apartment.

"I'm really proud of that program," Peter said. "But there are a few more things I still want to do with it."

"Better tell me — or show me — another time. Got to run. But thanks for having me over."

"My pleasure."

In a cab, Harris wondered at Peter's intense attachment to his electronic toy. It was all he would talk about in the office these days. And when you visited, and he showed you the computer and the games it played, it was as though he had a beautiful new girlfriend to introduce, or a clever child.

Harris caught the eleven o'clock train at Grand Central, and sat at the rear of the smoker; the last car, it would put him where he would want to be on the North White Plains platform. He lit a cigarette and settled back in his seat. He had done pretty well as the ruler of an interstellar empire — *after* he twice had been executed by rebels, and once had wrecked the economy so that a dark age had come.

Harris scarcely noticed that the train had stopped in the tunnel. Once he had figured out what was going on in the game, *then* he had done all right. He smiled at his reflection in the window. He'd be as wrapped up in it as Peter pretty soon, if he wasn't careful.

That was his next-to-last thought. The pain and the crashing sound came together. He didn't have time to think through to the conclusion that his train had been, was being, struck in the rear by another. His last thought was a wordless realization: that he was dying.

Watch this space next month for the exciting conclusion of our story.

Notes On dBase II

The BASIC-like command language within dBase II has some quirks which may cause confusion for a person used to Microsoft-like BASICs. A few of these are documented, most have to be discovered by trial and errors. Some examples:

A carriage return entered in response to the following command will give a string length of 1, and a value of a single space.

```
    Accept "give me a string" gstring
```

This makes it difficult to test for a null entry, since there are other entries which will return the same values.

But remember that if you get the string by

```
     store "             " to gstring
     @ 0,0 say "gimme a string" get gstring
```

a null entry will return the previous value of gstring, which in this case is 7 spaces.

In a typical application you may want to take names and addresses from a data base and write them to a MailMerge file for use with WordStar. Let's suppose that you have fields of 35 characters in length called name, address, city, state, zip.

So you write out the database to a delimited file:

```
    Copy to Mail.dat name,address,city,state,zip ;
         delimited with ' " '
```

Surprise! Since the field widths of name, address are 35 characters each, "Mail.dat" has a 35 character blank-filled field for each. MailMerge will print the spaces, making your letter unreadable. If you omit the "delimited" clause, the spaces will be ignored by MailMerge, but the data file will have a line length of 100+ characters and thus be difficult to view. You will also have to scrupulously avoid any commas in these fields.

So why not strip off the blanks? Easy in Microsoft BASIC:

```
10 SPOT = LEN(OLD$)
20 WHILE MID$(OLD$,SPOT,1)= 'B'
30     SPOT = SPOT-1
40 WEND
50 NEW$ = MID$(OLD$,1,SPOT)
```

Since in dBase

```
   @(<char string 1>,<char string 2>)
```

roughly corresponds to

```
   instr(string,searchstring)
```

and

```
   $(<char expression>,<start>,<length>)
```

corresponds to

```
   mid$(string,startpos,numofchars)
```

you might think

store $(oldstring$,1,@(oldstring$," ")) to newstring would work. The error, though, occurs on the first left paren.

What you have to do (don't ask why) is

```
   store @(oldstring,"    ") to pos
   store str(pos,2) to pos1
   store $(oldstring,1,'&pos1') to newstring
```

This allows you to bring the strings out of the database, edit them, and write them to disk with

```
   Set alternate to "MAIL.DAT"
   set alternate on

   ... blah blah ...

   Set alternate off
```

Another approach is to construct one line for each entry, by using the 'concatenation with blank squash' operator, and stripping the blanks off the right end of this large string just before writing it to the file.

```
   e.g.  store name-","-address-","-city-","-state-","-
         zipto recstring
   (strip blanks as above)
   (write to Mail.DAT)
```

On CP/M 2.25a For the TRS-80 Model II

Peachtree has developed a special LST driver for this CP/M — some dealers may have it. WordStar users can get best performance from CP/M 2.25a by installing as a TRS-80 II terminal with the next choice 'N' as in an FMG CP/M (NOT a 'Y' as stipulated for Lifeboat's!!). It's preferable if you have a serial printer to use WordStar's direct driver; install serial printers on the TRS-80 Model II by specifying Model II SIO on the driver menu and using the proper protocol within WordStar. If you are installing WordStar using the CP/M LST device, for printers needing a communications protocol (such as ETX-ACK or XON-XOFF), installation should be implemented in the CONFIG utility of CP/M 2.25a only, NOT in WordStar's INSTALL; tell it 'no protocol'.

A Review of Pascal MT+

by James Gagne

Version 5.2 of MT MicroSYSTEMS' native-code Pascal compiler, newly released by Michael Lehman, has already gained a reputation in the community as the best CP/M-based Pascal system. It features a native-code compiler that produces Microsoft compatible, relocatable machine code *directly* (no assembly required) and conforms to the new ISO draft Pascal standard. However, I was disappointed in the usefulness of /MT+ as a development system.

If one just examines its specifications, it is an extremely impressive system:

The compiler features separate compilation of code modules; in-line assembly code; chaining; full logical byte and 16-bit word operations; your choice of 32-bit floating point, 18-digit BCD, or AMD9511 hardware-supported real numbers; full compatibility with most UCSD Pascal extensions; capability of creating stand-alone, ROM-based microcomputer applications; direct access to input and output ports; and optional user-supplied error handlers. The best new feature for large programs is direct support of up to 256 memory overlays.

And that's just the compiler. In addition, there's a full symbolic debugger, a disassembler (which intermingles Pascal source with disassembled machine language for a super, machine-level listing), and a fast, memory-efficient linker that's Microsoft compatible (except for a few, rarely used features) and which also supports the 256 memory overlays.

The next new development is the speed-programming package. A screen-oriented editor forms the base, which you configure with a Pascal program for the terminal you are using. In addition, if you are familiar with Pascal, it is relatively easy to map the commands of the editor (control-a through control-z) to whatever function keys are present on your terminal. This editor combines the features of the original UCSD editor and WordMaster which best support Pascal source creation.

Once you've created or corrected a Pascal program, you can: a) automatically format your program, b) call a rapid syntax scanner to point out any syntax errors *while still in the editor*, where the information is maximally useful, and c) use a variable-name checker to tell you which names have occurred only once in your source and are therefore likely to have been undeclared or misspelled. Finally, a fast compiler (which omits the syntax check by assuming you've already done it) compiles your program.

Reviewer Biases

I come to the job of reviewing a Pascal compiler with a definite point of view. I've been a UCSD Pascal fanatic since I managed to get my first Z-80 UCSD system going in 1977.

I've used CP/M longer than the UCSD system and am quite familiar with it. But using high-level languages under CP/M is simply too slow. I can't stand BASIC and was never interested in FORTH.

Pascal is appealing because of its clarity and its elegance, and because of the ease with which software tools can be built and maintained. I've gradually built larger and larger programs in the UCSD system, and am in the process of releasing a 12,000-line (nearly 70K of p-code, 24K of data) text formatting program.

Therefore, I reviewed Pascal/MT+ paying particular attention to the ease with which one could create software tools with which to build large programs, and also to the support for separate compilation and memory management.

I have spent perhaps thirty hours with the /MT+ system, allowing me to become moderately familiar with its facilities. It would take another two months of use to describe the system from the level of real familiarity; it is that big.

Installation

Installing the compiler and linker consisted of copying the distribution disk. I found the instructions for modifying the editor for my terminal reasonably clear and straight-forward, although there are about 35 files involved (half of which allow you to link the whole thing together with a single SUBMIT file, which is supplied). Automatic linking of SPP's many modules went well; it was fun to see the prompts go whizzing by. I would guess that if you're familiar with Pascal and have a standard terminal, it would take thirty to sixty minutes to get going.

Using the Speedprogramming Package

The SpeedProgramming Package ("SPP") signs on with a menu that allows you your choice of tools, from listing a directory to executing a program (complete with separate prompts for the program name and run-time commands) to compiling, linking, formatting, or executing a syntax or variable check. All utilities work only on the program held in the SPP buffer, which is 16K bytes or so with a 64K system.

The directory lister consistently wrote garbage to my screen, and I ignored it. I tried writing and linking in one of my own, but I couldn't figure out how to trick CP/M into letting me read logical Groups 0 and 1, where the directory resides. (It's been done, and allows you not to make assumptions about disk format, which are required if you read physical sectors.)

The editor was reasonably quick, except for the darn disk delays imposed by CP/M. (Loading a 3-line program took ten seconds, twenty times longer than my UCSD system on the same computer.) Oops! SPP ignores lowercase commands, so I need my shift

(continued next

key. This is somewhat of a pain, since other commands require the control key. So I have to grope with both hands.

The SPP editor is line-oriented. Any lines longer than 80 characters are truncated without warning. (Watch out for insertions in the middle.) I can insert a carriage return anywhere with a control-N. But you can't erase a carriage return once inserted; you can delete only single characters within a line (control-G) or entire lines (control-Y). This is a pain. The delete function needs work.

Although there's no way to edit a file larger than the buffer, you should be breaking your stuff up into modules or $I (include) files anyway.

The editor fell apart when I edited OTHELLO, a test program I'd transported from a UCSD Pascal Users' Society library disk. The problem is that UCSD pads text with nulls, which my conversion utility does not adequately strip out. Upon loading this program into SPP, the editor added a lot of junk to the end of the buffer and ultimately crashed; the compiler was no more able to cope. And the system did not indicate what the problem was.

Editing text containing nulls is admittedly a severe test. But my other CP/M-based editor took it completely in stride, and I'm worried that you'd never know what happened if your disk system added some junk to the file you were editing.

Overall, I rate the robustness of the editor as poor. It works just fine so long as your programming practices match those of the author.

The syntax checker and Pascal source formatter worked well and were fast. When the syntax checker finds an error, it puts an error description at the bottom of the screen and positions the cursor at the offending spot. When faced with a portion of a program (e.g., the contents of an include file), the syntax checker complained that there was no PROGRAM declaration and would not go further. The for-tter provided a handsome program at designed both to be tolerant of ate syntax and to point it out s only problem is that if the is nearly full, it will trash nes of your program.

The variable-checker was of less use, simply writing to a file named "NAMES.$$$" all variables that were not Pascal reserved words and occurred only once. The problem is, I usually misspell my variables consistently. Luckily, the variable-checker ignores whether variables are upper or lower case or mixed, like the compiler, and should be a great help for the occasional typo.

Overall, the syntax and variable checks spotted about half my mistakes, with the rest pointed out much later in the last pass of the compiler.

How fast is SPP?

I found that SPP didn't save as much time as I'd hoped. The SPP fast compiler is only about 10% faster than the regular one for small programs. And it ignores $I (the include compiler directive) within the source, reading only from memory, and thus limiting program size to about 500 lines.

To cycle from the editor, through a compilation/linkage and program run and then back to the editor, takes about two minutes for a one-line program. For larger programs, add about a minute for every 150 lines. This is terrible, although in comparison with most of the other CP/M-based, machine language compilers, it is pretty good. (Compare with 15 seconds for my UCSD system, 35 seconds for Apple Pascal, or a little over a minute with Pascal/M.)

To compile and link when not using the SPP package involves a basic 90-second overhead. This is the time required to load the three passes of the compiler, create the intermediate file and then the relocatable file, and load and execute the linker. In addition to this baseline overhead, the system compiles and links at about 200 lines per minute, a respectable figure. These times would be reduced with a hard disk system.

Once the program itself has been created and linked, it runs about ten times faster than a comparable program running under UCSD Pascal and twenty times faster than Pascal/M. It is in turn nearly as fast as code produced by PL/I and half as fast as an equivalent program written in assembly language. Interestingly, it is about half as fast as the best 16-bit microprocessor Pascal, indicating that all you may need to keep up to speed is a fast Z-80. (These times are taken from the carefully wrought benchmark comparison by Jim Gilbreath, "A High-Level Language Benchmark", *Byte*, *6*:6, pp. 180-198, Sept. 1981.) (Note that the UCSD system has a new wrinkle: soon, a p-code to native machine code translator will convert time-sensitive portions of a program to provide a factor-of-10 speedup.)

Using the compiler

I found the user interface to the compiler, like to the rest of the system, to be superb. Prompts and error messages were explicit, and the program lets you know what's going on. Again, lower-case commands were disallowed. And error messages occurring during the final pass were inconvenient to find and correct, since you were given only the procedure name, error number, line number from the beginning of the file, and the last identifier name. I'd like in addition a descriptive error message as well as line displacement from the start of the procedure.

Once when it had been correctly edited, at first my 787-line OTHELLO program would not compile, due to lack of memory in my 56-K system. Then I tried the suggestion in the manual of recovering stack space occupied by rarely used built-in procedures, using the $K compiler directive. Now the file compiled. But the linker ran out of memory until I specified the data area with a "/D" linker directive, which tells the linker not to initialize the data area and thus allows more code to be linked. But then I could not get the thing to run; invoking OTHELLO from CP/M simply caused a return to the operating system. Much playing around on my own was not helpful. Sigh. A call to MTMicroSYSTEMS support provided the answer: there was an undeclared call to "@XOP" pointed out by the linker. It turns out that this is a real number procedure, and you must link in your choice of real number libraries. Otherwise a call to @XOP is left as a call to 0: a warm boot.

It is not clear what limitations one would face with a bigger program in a 64K system. However, it does appear that much over 1000 lines, you would be forced to use separate compilation and overlay modules, or the thing won't compile or link.

The first pass of the compiler produces the listing if you desire. The listing is simply the source code with line numbers, a nesting level indication, and INCLUDEd files pulled in. I found it of use only when debugging, when you need line numbers to find out where you are. To obtain code sizes of each procedure, you must have the printer following the console output when the compiler is operating, since the last pass produces a list of procedure names with relative offsets from the start of the program. The only other way to find your way around your code in memory is to opt for a symbol table dump while linking (but it's in the wrong order) or to use the disassembler.

The disassembler is very slick, producing a listing of intermingled Pascal source lines and their accompanying assembly code. (You'd have to comment out the Pascal source to reassemble, however; its purpose is as a listing.) The disassembler produces garbage unless you choose the disassembly option while compiling; this option adds considerable information to the relocatable code to assist the disassembler in its work. It is intended for those situations when you really must dig around in your code, and is not really for routine use while creating a program. The run-time library is not included in a disassembled listing.

The debugger allows you to examine variables and to execute a program in trace mode (one or more lines at a time, traced by line number) or at full speed under breakpoint control (set at procedure/function entry only). You may enable a display of procedures being entered and left. Its use is limited by the fact that you can't change variables. Nor is it possible to begin execution anywhere but the beginning of a program or at a spot where a breakpoint had been encountered. Digging through arrays, records, and other structured variables is entirely up to the user; all you're given is a DDT-style Hex/ASCII dump of the 16 bytes of your choice (offsets from the start of the variable are permitted). Although a SID-compatible symbol table is available from the linker, it lists only procedure entry points, not variable locations, and is of limited help.

Nevertheless, the debugger is fully symbolic and is an enormous improvement over inserting those darn "..in procedure xyz; abc = 13" trace messages in program source. Its use requires a listing and compiling with the debugger option set, then linking in the debugger itself, which becomes the main program. In trying to debug my OTHELLO program, I found that adding the debugger unquestionably exceeded memory bounds; it is of limited use in large programs. And the debugger option increased the size of one big procedure past the /MT+ upper limit (2 kilobytes per procedure body).

Support for software tools

I am enormously fond of creating procedures and functions that perform a simple task well (say, reading an integer from the console in an uncrashable manner) and can be used anywhere, building upon each other. To be useful, these "software tools" (see the excellent text of the same name by Kernighan and Plauger) must make no assumptions about their environment and be as general as possible. This means that all communications are via their parameter list; no global variables are used. Because these tools tend to be used frequently in a program, two or more invocations may be active at one time. Thus, they require an environment that supports recursion, where each call of the procedure creates a separate data area on the program stack.

Pascal/MT+ has its roots in Pascal/MT, which was a subset of Pascal designed to replace assembly language in real-time control applications, a task it performed admirably. In this environment, the time and code space overhead of recursion was unnecessary, and Pascal/MT forbade recursion unless you set a compiler option to enable it. Moreover, type and range checking had to be explicitly turned on, since they were often a nuisance.

Unfortunately, these practices persist. I guess it's difficult to explain to a non-Pascaler why defaulting type and range checking to off is like the default state of your auto accelerator being full throttle; suffice it to say that it is not a way to encourage program reliability. Moreover, the /MT+ recursive environment lacks one important feature: a reliable way to handle exceptions. While the GOTO works fine across procedures in a nonrecursive environment, Pascal/MT's stack discipline does not support a reliable GOTO when it is operating recursively, giving you no way to get out quickly from a large nest of procedures when an error or exception occurs. This violates the ISO standard and is a serious flaw. (There is an EXIT procedure, but this only gets you out of the current procedure, and it also does not work in recursive mode.)

Separate compilation

With version I.5, UCSD Pascal introduced an extraordinary mechanism for developing software tools, the UNIT. A UNIT is a separately compiled module intended for use by other programs, which contains an explicit, public INTERFACE section followed by an IMPLEMENTATION portion that is private and can be modified without disturbing the linkage to the outside world. When a UNIT is declared in a host program, the compiler automatically reads in the INTERFACE and treats it as if it were part of the host program declaration section, saving oodles of typing and providing automatic type checking. Thus, not only can UNITs be maintained independently of their host programs, they can be sold to others in object form as software tools.

Separate compilation is also supported under Pascal/MT, under the syntax:

```
    MODULE <name>;
<label, const, type, var declarations>
<procedures and functions>;
    MODEND.
```

The $E- compiler option allows you to declare portions of your code as private and unavailable to the linker. Unfortunately, if you use variables or procedures/functions from other modules or programs, they must be declared EXTERNAL in the host program, and no type checking or parameter list verification is done. Thus, if you are compiling a large program in

(continued next page)

pieces and want each part to have access to your global variables, you have extra typing and praying (that you've used variable types accurately) to do.

The suggestion from MTMicroSYSTEMS was to put global types in a separate INCLUDE file (since types do not have to be declared as EXTERNAL), then a second file containing a re-edited version of global variables with EXTERNAL added. EXTERNAL procedures and functions might be added to yet another INCLUDE file, since any EXTERNAL procedure and function declarations not used are simply ignored, although they consume symbol table space.

Suggestion to improve the present syntax: let the compiler read the real global declarations under the influence of a compiler option telling it that these names are all external.

Program overlays and memory management

One of the nicest new features of version 5.25 is direct support of program overlays. You can declare up to 16 areas in memory, each of which can contain up to 16 overlays, for a total of 256 possible program segments. Moreover, when you set up the linker to include the run-time library, it keeps track of which run-time modules are present in the host program and does not add them to overlay modules. And /MT+ keeps track of which modules are in memory, so that if one module is called repeatedly, it is not reloaded each time.

Not bad.

Because 8080/Z-80 processors do not readily allow location-independent machine code, you must manually figure out where each portion of the program (code and data for each module) is to go. This is done by compiling each module, then test linking it to determine code and data size. Then set up your memory map, and relink each module to its actual location.

Documentation

I found Pascal/MT+ to be clearly explained in its accompanying manuals. The prose was occasionally clumsy and contained grammatical errors, but I could always find the information I needed quickly. One oversight: the names of run-time procedures and functions are provided for your use, but no parameters or function types are provided, making them useless. The /MT+ manuals assume familiarity with Pascal and could not be used by themselves if you're a beginner.

Conclusion

Writing a compiler is always an enormous task, but trying to squeeze native-code Pascal into a 64K machine is stupendous. Mike Lehman reports that the code generated by the compiler has consistently worked well; I have no reason to doubt him He is to be commended for a superb effort.

Clearly, the task of making the Pascal/MT+ system truly robust still lies ahead. For now, the user will have get used to the usual idiosyncracies, a victim of an impressive array of facilities that work well if you program in the manner of the author.

I prefer the UCSD system, particularly the factor-of-ten speed advantage afforded during compilation. Even Apple Pascal allows compile/link/run/edit phases four times faster than /MT. Among CP/M-based Pascals, however, Pascal/MT+ offers respectable compilation times along with rapid program execution (provided your program isn't i/o bound, which is usually the case). The Speed-Programming Package will assist you in writing small-to medium-size programs and is of less use with larger programs, although you can run program modules through the syntax and variable checkers if they are syntactically complete.

The SPP fast compiler would be greatly aided by being reconstructed as a one-pass compiler optimized for speed. I wouldn't mind recompiling programs that I knew worked with a slower compiler in order to obtain smaller object files or faster execution. However, I suspect that speed will have to await 16-bit micros with larger address spaces.

CPMUG NEWS

CPMUG wishes to congratulate the SIG/M Users Group for their contribution to users of public domain software throughout the world.

In a spirit of cooperation, SIG/M has endorsed the incorporation of their volumes into CPMUG.

Welcome to all SB-80 Users: you may now take advantage of the entire CPMUG Library of 64 volumes.

Ordering From CPMUG

CP/M Users Group library diskettes are available from The CP/M Users Group, 1651 Third Avenue, New York, New York 10028. As of this date, the Library contains 64 volumes of software available on 8" IBM single-density CP/M diskettes, or on North Star diskettes readable by users of double-density CP/M 1.4, double-density CP/M 2.2, quad capacity CP/M 2.2.

The complete CPMUG catalogue is available for $6 prepaid to the U.S., Canada and Mexico. The cost to all other countries is $11 prepaid. Members receiving the material are reminded that software contributions are necessary if the exchange program is to prosper. Software contributions are gladly received for inclusion into the Library with the understanding that the contributor is authorized to make the material available to others for their individual non-commercial use.

Please write for more information to CPMUG at the address above.

Did your subscription start last November or December? See page 39.

CPMUG Catalogues: Volumes 55, 56, 57, 58, 59, 60, 61, 62, 63, and 64

CPMUG VOLUME 55

Original Adventure run time implemented for CP/M

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.055	CONTENTS OF CPMUG VOL. 55
		ABSTRACT.055	NOTES ON ADVENTURE
		UGFORM.LIB	SUBMITTAL FORM
		SIG/M.LIB	SUBMITTAL FORM
55.1	39K	AD.COM	
55.2	63K	ADVENTUR.MSG	
55.3	2K	ATAB.DAT	
55.4	4K	COMMON.DAT	
55.5	1K	KTAB.DAT	
55.6	1K	LTEXT.DAT	
55.7	1K	RTEXT.DAT	
55.8	1K	STEXT.DAT	
55.9	5K	TRAVEL.DAT	

CPMUG VOLUME 56

Original Adventure source code implemented for CP/M

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.056	CONTENTS OF CPMUG VOLUME 56
		ABSTRACT.056	IMPLEMENTATION NOTES
		UGFORM.LIB	SUBMITTAL FORM
		SIG/M.LIB	SUBMITTAL FORM
56.1	28K	ADINIT.COM	
56.2	13K	ADVENSUB.FOR	
56.3	5K	ADVENT.FOR	
56.4	74K	ADVENTUR.DAT	
56.5	29K	ADVINIT3.FOR	
56.6	47K	ADVMAIN.FOR	
56.7	5K	INSUB.FOR	
56.8	5K	MAINSB.FOR	

CPMUG VOLUME 57

Expanded ADVENTURE 8080/Z80 version supercedes SIG/M volume 3. This version will sense the type of processor (Z80 or 8080) and take advantage of Z80 instruction set if available.

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.057	CONTENTS OF CPMUG VOL. 57
		ABSTRACT.057	NOTES ON EXPANDED ADVENTURE
		UGFORM.LIB	SUBMITTAL FORM
		SIG/M.LIB	SUBMITTAL FORM
57.1	36K	ADV.COM	
57.2	105K	ADVT.DAT	
57.3	31K	ADVI.DAT	
57.4		4K	ADVI.PTR
57.5	15K	ADVT.PTR	

(continued next page)

CPMUG VOLUME 58

Miscellaneous CP/M Utilities

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.058	CONTENTS OF CPMUG VOL. 58
		UGFORM.LIB	SUBMITTAL FORM
		SIG/M.LIB	SUBMITTAL FORM
58.1	60K	3740UTIL.ASM	Copy CP/M to and from 3740 format
58.2	8K	3740UTIL.DOC	Copy CP/M to and from 3740 format
58.3	2K	BDOS-PAT.ASM	Make user 0 (CP/M 2.X) public
58.4	7K	BMAP7/11.ASM	Print allocation map
		** SOURCE **	CPMUG VOL. 47 (A.K.A. FMAP7/11.ASM on some disks; otherwise -identical
58.5	1K	CCPPATCH.ASM	Make drive A: default for .COM files
58.6	4K	CHANGE.ASM	ALS-8 to CP/M converter (DR. DOBBS)
58.7	8K	CRCK3.ASM	CRC check on a file
		** SOURCE **	CPMUG VOL. 46 - identical.
58.8	5K	DIRFIX.ASM	Rids attribute bits for 1.4 compatibility
58.9	9K	DIRSIO/1.ASM	Sorted DIR with SYS and MP/M options
58.10	1K	DOWHILES.LIB	Macro lib. used by 58.1
58.11	16K	DU-8/12.ASM	Update of CPM USER GROUP 40.20
		** SOURCE **	CPMUG VOL. 46 - identical.
58.12	8K	DUPUSR2.ASM	Create duplicate directory entries w/new user #
58.13	4K	EQUATES.LIB	Macro lib. used by 58.1
58.14	2K	FILPRINT.ASM	Turn .TXT into .COM
58.15	27K	FINDBD37.ASM	Update of INTERFACE prgm to lock out bad blocks
58.16	9K	FMAP6/12.ASM	Update of CP/M USER GROUP 40.24
58.17	16K	MACROS.LIB	Macro lib. used by 58.1
58.18	2K	NCOMPARE.LIB	Macro lib. used by 58.1
58.19	1K	PG0EQU.ASM	Part of 58.9 DIRSIO/1.ASM
58.20	2K	SELECTS.LIB	Macro lib. used by 58.1
58.21	1K	SURVEY.COM	List disk, memory use, and other parts
		** SOURCE **	CPMUG VOL. 46 - almost identical (1 byte off).
58.22	13K	SURVEY3.ASM	List disk, memory use, and other parts
		** SOURCE **	CPMUG VOL. 46 - conditional assembly on this disk I.M.S. controller; version on VOL. 46 selects the TARBELL controller.
58.23	1K	SYMSTACK.LIB	Macro lib. used by 58.1
58.24	1K	WHENS.LIB	Macro lib. used by 58.1
58.25	8K	XDIR6/28.ASM	Sorted directory with sizes

CPMUG VOLUME 59

8080/8085, Memory, iCOM Disk Controller diagnostics

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.059	CONTENTS OF CPMUG VOLUME 59
		UGFORM.LIB	SUBMITTAL FORM
		SIG/M.LIB	SUBMITTAL FORM
59.1	14K	CPUDIAG.ASM	CPU test diagnostics for 8080 and 8085
59.2	2K	CPUDIAG.DOC	CPU test diagnostics for 8080 and 8085
59.3	12K	MEMDIAG.DOC	Memory diagnotics
59.4	34K	MEMDIAG.ASM	Memory diagnostics
59.5	73K	3712DIAG.ASM	iCOM single density disk controller diagnostic
59.6	93K	3812DIAG.ASM	iCOM double density disk controller diagnostic
59.7	2K	3X12DIAG.DOC	iCOM disk controller diagnostics
59.8	4K	CP/M-NET.MSG	Proposed networking of user groups

CPMUG VOLUME 60

6502 Simulator- Reference Dr. Dobbs 8/80
6502 Zapple monitor

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.060	CONTENTS OF CPMUG VOL. 60
		ABSTRACT.060	DESCRIPTION OF EACH MODULE IN THE 6502 SIMULATOR
		UGFORM.LIB	***** NO SPACE -AVAILABLE ***** ***** ON VOLUMES 55 - 59.*****
60.1	11K	ZILASM.COM	6502 Simulator
60.2	5K	ZX65.COM	
60.3	98K	ZX65R.PRN	
60.4	39K	ZX65R.ZSM	
60.5	15K	ZXART.DOC	
60.6	9K	ZXHINTS.DOC	
60.7	23K	ZXLDR.PRN	
60.8	7K	ZXLDR.ZSM	
60.9	1K	ZXTAB1.DOC	
60.10	1K	ZXTAB2.DOC	
60.11	1K	ZXTAB3.DOC	
60.12	1K	ZXTAB4.DOC	
60.13	2K	CONCOD.LIB	Control codes for 6502 Zapple
60.14	1K	ZAPMON.MOS	Zapple monitor for 6502
60.15	5K	ZAPMON.HEX	
60.16	11K	ZAP3.INS	
60.17	3K	ZAP5.INS	
60.18	3K	ZAP1.INS	

CPMUG VOLUME 61

Bulletin Board Related Software System
File transfer utilities
CP/M utilities

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.061	CONTENTS OF CPMUG VOLUME 61
		UGFORM.LIB	SUBMITTAL FORM
		SIG/M.LIB	SUBMITTAL FORM
			BULLETIN BOARD RELATED SOFTWARE
61.1	23K	DCHBYE55.ASM	Remote console for DC Hayes Modem
61.2	3K	FLIP-8/8.ASM	Switch remote console to originate mode
61.3	5K	TAG.ASM	Set F1 bit
61.4	9K	MLIST34.ASM	Type command w/16K buffer
61.5	45K	MODEM926.ASM	Update of CP/M User Group 40.28
61.6	21K	PLINK925.ASM	Update of CP/M User Group 19.4
61.7	2K	USER-8/8.ASM	Replaces CP/M User CMD on remote CPU
61.8	20K	XMODEM32.ASM	DC Hayes support

(continued next page)

CPMUG VOLUME 61 (continued)

NUMBER	SIZE	NAME	COMMENTS
61.9	4K	PURGE.ASC	Requires MBASIC
61.10	20K	RIBBS.ASC	Requires MBASIC
			FILE TRANSFER UTILITIES
61.11	7K	MIC-XFER.ASM	Data transfer between Micropolis CP/M 1.4 and 8″ systems
61.12	7K	MIC-XFER.DOC	Data transfer documentation
61.13	3K	XFER5-8.ASM	Transfer files between 5″ and 8″
61.14	3K	XFER8-5.ASM	Transfer files between 5″ and 8″
61.15	5K	V2FORMAT.ASM	Versafloppy system
			CP/M UTILITIES
61.16	9K	MENU.ASM	Creates menu of all .COM and .BAS files
61.17	5K	MEM-MAP.ASM	Use to map RAM/ROM
61.18	8K	MOVE6/12.ASM	Single drive copy program
61.19	8K	RELDUMP.ASM	Dump Microsoft .REL files
61.20	1K	SAP-FIX.DOC	Patches for CP/M User Group Vol. 19.8
61.21	5K	TEXCLEAN.ASM	Clear bit 7 of a test file
61.22	2K	TPA3.ASM	Computes size of TPA
61.23	3K	Z80EXT.LIB	Extra Z80 Opcodes

CPMUG VOLUME 62

PASCAL and Communications related programs

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.062	CONTENTS OF CPMUG VOLUME 62
			PASCAL RELATED PROGRAMS
62.01	1K	BOOT.ASM	Sample BOOT for PASCAL
62.02	3K	BOOTER.DOC	PASCAL documentation
62.03	7K	TEST.ASM	Scan and load 8080/Z80 Pascal interpreter
62.04	8K	PAS2CPM.ASM	Format conversion
62.05	7K	PASCAL.ASM	Initialization of PASCAL system
62.06	17K	PASCAL.COM	
62.07	1K	PASCAL.DOC	
62.08	12K	PASTOCPM	Format conversion
62.09	1K	PBOOT.ASM	Utilities to initialize PASCAL system
62.10	4K	PGEN.ASM	
62.11	1K	PGEN.COM	
62.12	6K	PINIT.ASM	
62.13	2K	READ.ME	PASCAL startup documentation
			COMMUNICATIONS RELATED PROGRAMS
62.14	5K	CHAT13.ASM	2-way communications with remote caller
62.15	24K	DCHBYE57.ASM	Upgrade of remote console DCHBYE55 on CPMUG 61.01
62.16	46K	MODEM5A.ASM	Auto-dial and auto re-dial capability on DC Hayes and PMMI modem boards
62.17	21K	PLNK1018.ASM	Upgrade of PLNK925 on CPMUG 61.6 and CPMUG 19.4 with more modem types
62.18	4K	NEWBAUD.ASM	Switch BAUD rate on a remote PMMI to readjust modem speed
62.19	20K	RBBS22.ASC	Update of RBBS in CPMUG volume 61
62.20	6K	RBBS22.DOC	
62.21	9K	RBSUTL22.ASC	
62.22	8K	MBOOT3.ASM	Compacted version of MODEM for RECV only usage
62.23	24K	XMODEM38.ASM	Update of CPMUG 61.8 - remote CP/M to CP/M transfer

CPMUG VOLUME 63

NUMBER	SIZE	NAME	COMMENTS
		-CATALOG.063	CONTENTS OF CPMUG VOLUME 63
		UGFORM.LIB	SUBMITTAL FORM
		SIG/M.LIB	SUBMITTAL FORM
63.01	2K	AUTOX.ASM	Forces a CP/M command from a user level
63.02	10K	CPYFIL15.ASM	Copy large files greater than 512K through PIP utility
63.03	11K	CRCK10/6.ASM	Upgrade of CRCK3 in CPMUG 58.7
63.04	9K	DIRS1015.ASM	Sorted directory from DIRS10/1 in CPMUG 58.9
63.05	3K	DISPLAY.COM	Similar to DIR/ED.COM using display commands only
63.06	3K	DISPLAY.DOC	
63.07	3K	DISPLAYP.ASM	
63.08	18K	DU-10/26.ASM	Update of disk utility in CPMUG 58.9
63.09	10K	FIND3/18.ASM	Multiple file search routine
63.10	29	FINDBD38.ASM	Update to FINDBD37 for locating bad blocks of disk space in CPMUG 58.14
63.11	1K	LISTGRPS.ASC	List track and sector assignment for each group
63.12	10K	MDIR8/17.ASM	Master directory by users in alphabetic sequence
63.13	17K	MFT45.ASM	Dr. Dobbs single drive multi-file transfer program
63.14	7K	MICXFER.ASM	Micropolis and regular CP/M file transfer
63.15	5K	MICXFER.DOC	
63.16	9K	NFMAP.ASM	Sorted directory with option of writing file of names
63.17	14K	NLIST.ASM	Lists disk file on LST: device
63.18	17K	SD-12/15.ASM	Sorted directory with sizes
63.19	10K	SECTOR.ASM	Sector disk maintenance program
63.20	2K	SHOWGRP.ASC	Print track and sector addresses of groups
63.21	2K	TERMTEST.ASM	Terminal diagnostic program
63.22	3K	USERLST.ASM	Patch for displaying current user level within CP/M prompt
63.23	8K	VLIST11.ASM	Variable speed TYPE routine
63.24	4K	WHICH.ASM	Displays present CP/M release level
63.25	8K	MIKEBIOS.ASM	Flash Writer I/O driver

CPMUG VOLUME 64

Games, disassembler, North Star Basic patch for CP/M, CDOS simulator, and other utilities.

NUMBER	SIZE	NAME	COMMENTS
		CATALOG.064	CONTENTS OF CPMUG VOLUME 64
		UGFORM.LIB	SUBMITTAL FORM
		SIG/M.LIB	SUBMITTAL FORM
64.01	6K	LANES.BAS	Games using MicroSoft Basic
64.02	2K	JOURNAL.BAS	
64.03	15K	GAMMONB.BAS	
64.04	3K	BLACKBOX.BAS	
64.05	1K	BOGGLE.BAS	
64.06	11K	GAMMON.H19	Heath terminal adaptation
64.07	8K	NIM1.H19	
64.08	5K	OTHELLO.H19	
64.09	12K	STARWAR2.H19	
64.10	17K	ALS8CPM.ASM	Converts Processor Tech assembler files to CP/M format
64.11	1K	DIV16B.ASM	16 bit division by 8 bit divisor
64.12	5K	DSIM.LIB	CDOS calls for CP/M
64.13	22K	NSCPM48.ASM	Patches to run North Star Basic under CP/M
64.14	8K	NSCPM48.COM	
64.15	1K	RETURN48.COM	
64.16	33K	REZ.ASM	RESOURCE resurrected with .ASM
64.17	7K	REZ.COM	A nifty 8080 disassembler
64.18	26K	REZ.DOC	Modified for Z80 TDL op codes
64.19	36K	REZ.Z80	Modified for Zilog op codes

Still More Random Numbers

Editor's Note: Two more letters have arrived on the subject of random numbers.

October 6, 1981

To the Editor:

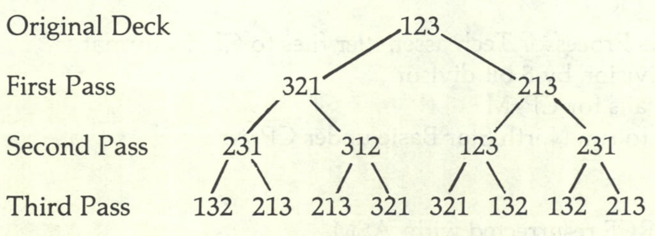
Developing correct and efficient algorithms for random number generation and their use are among the most difficult of computing problems. Their difficulty lies in that such algorithms are difficult to test. Many algorithms for the generation and use of random numbers appear intuitively correct but are, in fact, far off the mark. Recent examples of this have appeared in *Lifelines.* I hope to correct one of these.

In the October 1981 issue, the article by Bill Burton, "Better Random Numbers," contains a reasonable, but incorrect routine for shuffling a deck of cards. That this algorithm is incorrect can be seen easily by considering an example of a deck with three cards (numbered 1, 2, and 3). The subroutine for shuffing this deck becomes

```
540  FOR  I=1  TO3
550       J=INT(RND*3)+1
560         IF  I=J  THEN  550
570       SWAP  A(I), A(J)
580  NEXT  I
590  RETURN
```

The function RND is the random number generator from BASIC-80 and returns a number in the range 0<RND<1, and the vector A contains the card numbers.

The actual action for this subroutine is diagrammed below. On the first pass through the FOR-loop, there are two possible outcomes — 321 or 213. On the second pass, 321 can become 231 or 312, and 213 can become 123 or 231, so the second pass can produce 231, 312, 123, or 231. The third and final pass has eight possible outcomes: 132, 213, 213, 321, 321, 132, 132, or 213. Combining the duplicate outcomes, this becomes 3 of 132, 3 of 213, and 2 of 321, each equally likely. Therefore, 37.5% of the decks shuffled will have the order 132, a like amount will have the order 132, and the remaining 25% will have the order 321. The orders 123, 321, or 231 are not possible. If we choose the "top card," we are not as likely to choose a 3 as either of the other two cards.



For a deck of 4 cards (1, 2, 3, 4), there are 24 theoretically possible shufflings. Only 12 are actually possible using this algorithm, with the ordering 2143 the most likely (11/81 or approximately 13.58%). This same pattern will hold true for all deck sizes. A necessary, but not sufficient, condition for a correct algorithm is that the number of possible orderings must be a multiple of the number of theoretically possible orderings generated. For n cards, the number of possible orderings generated by this algorithm is $(n-1)^n$, which is not a multiple of the number of theoretically possible orderings (n!). This automatically excludes the present algorithm.

The place where the author made his mistake was in point #2 of his description of an efficient shuffler — "The position of each card must change at least once." There is no reason why a card cannot be left in its original order. However, simply eliminating the IF-statement (line 560) does not make the algorithm correct — although it does make all shuffled decks possible, some are more likely than others. This algorithm has n^n possible orderings, which still is not a multiple of n!.

One solution (perhaps not the best) is to modify a correct algorithm to be efficient. Consider how you might shuffle the deck if you are given a list of random numbers and are not allowed physically to shuffle the deck. The algorithm might look like:

1) Number the unshuffled deck in any order
2) Choose a card at random from the unshuffled deck and put it in the shuffled pile.
3) The unshuffled deck is reduced by 1. Repeat 2) until no cards remain.

An example with a deck of four is diagrammed below:

Unshuffled	1234	124	24	2	
Shuffled		3	31	314	3142
	Original Deck	First Pass	Second Pass	Third Pass	Final Pass

While this requires two storage areas, unshuffled and shuffled, we can see that as the unshuffled deck is decreased, the shuffled deck is increased. This suggests that we might be able to use the same storage area for both decks. An example might look like:

```
Deck |1234     3|214    31|24    314|  2    3142|
   S |  U      S|  U    S |  U    S |  U    S   | U
    Original    First    Second    Third    Final
      Deck      Pass      Pass     Pass     Pass
```

S=Shuffled Deck   U=Unshuffled Deck

At the first pass 3 is swapped with 1. This is OK since the order of the unshuffled deck does not affect the shuffling process. This has the effect of leaving all the unshuffled cards in the upper part of storage.

For a deck of 52 cards the subroutine becomes

```
540  FOR  I = 1   TO 5 1
550       J = I N T ( R N D * ( 5 3 - I ) ) + I
560       S W A P   A ( I ) ,   A ( J )
570  N E X T   I
580  R E T U R N
```

Notice that we need to go through the loop only 51 times (one less than the number of cards in the deck). This is because when we reach the last card there is only one choice. The program is shorter and will execute slightly faster than the incorrect code given by Bill Burton. Notice that the number of possible orderings is n!, which is equal to the number of theoretically possible orderings.

The moral of this story is that extra care must be used in designing and testing algorithms involving random number generators.

Sincerely,
Ted H. Emigh
Assistant Professor of Genetics and Statistics
North Carolina State University

Sept. 3, 1981

To the Editor:

I did miss the intent of James R. Reinders' assembly language program for random numbers as explained in the answer to my letter last month. However, I must explain that I feel very confident in my ability to do most anything in MBASIC but also feel relatively inadequate in machine language. This also appears to be the position of the majority of microcomputer software hackers. When a routine will not take a measurable amount of operating time in the higher level language, I think that is the way to go.

Consequently, I took another look and refined my routines to allow the kind of randomizing that would be required by the long running program needing several random numbers.

The programmer knows the total number of random numbers he will require in a program. I have called this variable RAND.TTL. In the interpreter, he can dimension the array dynamically but if using BASCOM, he will have to dimension with an integer value. The first GOSUB sets up a list of randomly chosen seeds for the RANDOMIZE function.

Line 200 will get a random number and increment the list index. The next time you need a random number, GOSUB 200 and you randomize again with a new seed...and on, and on, and on up to the number selected at RAND.TTL.

Line 130 is a REPEAT-UNTIL loop for MB80 which loops forever looking for the space bar.

Bob Kowitt
The Computer Consultant

```
10  RAND.TTL=10   :  DIM RAND.NO(RAND.TTL)
20  GOSUB 120                     '           get enough random seeds
30  L=1 : GOSUB 200               '           get first random number
40  '  more program
50  GOSUB 200                     '           get another
60  '  more program
70  GOSUB 200                     '           get another
75      '        etc, etc, etc  up to RAND.TTL
90  '
100 END
110             ************************************
120 N=RND:PRINT:PRINT
130 FOR K=0 TO -1 STEP -1:K=(INKEY$=" ") :GOSUB 170 : NEXT
140 FOR K=1 TO RAND.TTL : GOSUB !&) : RAND.NO(K)=INT(N) : NEXT
150             RETURN
160 '           ++++++++++++++++++++++++++++++++++++
170 N=N*2 : IF N>32000 THEN N=N/3
180     RETURN
190 '           ++++++++++++++++++++++++++++++++++++
200 RANDOMIZE(RAND.NO(L)) : PRINT RND : L=L+1 : RETURN
```

Notice

Letters, tips and articles are welcomed from all our readers, but we particularly appreciate them when they arrive in machine-readable form.

Unless the sender specifies otherwise, we assume that all material submitted is intended for publication and that its transmission to us constitutes permission to publish. Unless other arrangements are made, any such material becomes the property of Lifelines Publishing Corporation.

Product Status Reports

New Versions

MDBS III
by Micro Data Base Systems, Inc.
This major update features improvements in speed, portability, flexibility and physical storage control. A proprietary access algorithm increases access speed for large sorted sets in moderate to large data bases.

A multi-user option has been added, permitting several users to access a data base concurrently. Record lockout is managed by the system, passively and actively. MDBS III is configured to run with CP/M-compatible MP/M and UNIX-type operating systems.

This system was developed in two forms; one is an assembler implementation designed for microprocessors (8080/8085, Z80 and others). The other is written in C, for larger systems with less memory constraint. Applications packages developed using the first form of MDBS III can be transported to 16-bit micros and some minis, including the PDP-11.

Physical placement of records can be declared for up to sixteen physically distinct data base areas. A designer can assign occurrences of a record type to one or more areas, and can declare how records are to be placed within an area or areas. A clustering option permits a group of records which are related (via a set) to be accessed more speedily.

Selective linking is supported, depending on the host language, so that only the routines needed by an application program are resident in main memory. This frees room for page buffers.

Data is maintained in a common internal representation regardless of the host language or languages used during data base creation and modification. Any available host language can be used for read or write access to a given data base.

Automatic multi-item sorting is supported for any set. The sorting can be declared in mixtures of ascending and descending order.

QRS, RTL, and DMU are available as add-ons to MDBS III. QRS now has a language syntax consistent with those of earlier QRS versions. Sold with QRS will be a separate program supporting interactive DML, because the initial version of QRS will not include these features. The Design Modification Utility provides utilities to aid modification of schemas. RTL now provides automatic logging-in for multi-user environments.

Data item specifications allowed in earlier versions are permitted in this one, except for logical and depending-on items. Logical items can be replaced by unsigned items, and string and binary items of variable length replace depending-on items. Data types allowed are: integer, unsigned integer, binary, real, internal decimal, character, string, time and date. Several types are automatically compressed. Up to 65,535 items per record type are permitted; up to 255 record types are allowed per data base schema. Recursive and multiple owner/member sets are supported, and fixed set retention can be declared. Multiple ascending and descending sort keys per set are permitted; sorting can be performed on on the record type names and on various data items.

User passwords are part of a data security system which includes data encryption. Instead of access levels,any combination of 16 access codes can be specified, providing up to 65,535 different codes which can be assigned to a user, data item, set, or area.

Some additional commands have been added to the Data Manipulation Language (DML) including commands to set currency indicators to null, and to set user definable currency indicators. The latter are intended for traversing a complex data base structure.

Boolean operations are now supported, with AND (for set intersection) and XOR operations. These are useful for obtaining record occurrences satisfying two or more selection criteria.

MP/M II
by Digital Research, Inc.

MP/M II is a new version of the multi-user operating system, designed for microcomputers with 8080/8085 or Z80 processors. The modular design includes a Terminal Message Processor (TMP) to read the user's command line, a Command Line Interpreter (CLI) for loading programs, BDOS, XDOS, and XIOS (EXtended Input/Output System). MP/M resides in less than 26K of memory, 13K of which must be common to all users. Total size is determined by the number and type of peripherals supported, and the resident system processes included when the system is generated.

MP/M II provides upward compatibility with CP/M, date and time stamps, password protection, 16 logical drives managing up to 512 megabytes, error handling, multiple printer support, CP/NET compatibility. It can manage up to 400K bytes of RAM, with 16K of common memory required for the nucleus of the operating system.

PAS-3 Medical
Version 1.77

This version is intended to fix all known bugs and has improvements in presentation of data on the screen. It permits the user to specify which drives the program and data will be on. This allows the system to be used with the Altos hard disk and other systems that have their Hard Disk starting at a drive other than A:.

Plink
Version 3.28
The bugs listed below have been repaired with this new version:

1. When the blank common .BLNK. was zero bytes long and contained a symbol, error #30 would result (e.g. see example 3 in Appendix D of the manual). This problem has been corrected.
2. The /ACTUAL option did not work when used with Microsoft format files; this has been corrected. Three new error numbers have been added:
   63-Microsoft Fixup error (indicates Plink-I or compiler bug).
   64-Common blocks were overlapped in memory by the /LOCATE option.
   65-Too many common blocks in module (maximum is 251).
3. Pascal MT files may now be linked.
4. Plink normally places some code at the front of the program to set the Z80 stack pointer to the top of available memory. This code now uses only 8080 instructions so that programs may be generated for these machines.
5. Plink now saves the X and Y registers when calling the operating system: some CP/M look-alikes smash these registers when called by a program.
6. Some versions of the Digital Research PL/I-80 compiler generate invalid .REL files when an empty string is declared: 256 garbage bytes are written to the .DATA. segment causing the linkage editor to overwrite the memory assigned to the following segment. This usually doesn't matter since the next segment is typically from a module that hasn't been loaded yet, but the problem may surface if the /LOCATE option is used, causing segments to be allocated out of order. Symptoms of this problem are fixup errors or smashed code in the program.
7. Another PL/I-80 problem has been fixed: modules having no code in them are now handled correctly. These would some times cause fixup errors or missing code or data in the output file.
8. Common blocks were not handled properly if they had the same name as a module: each instance of the common was assigned a different memory area. This has been remedied. Microsoft apparently handled this problem in their FORTRAN by setting the high order bit in the first character of each common block name. Unfortunately, this means that Microsoft FORTRAN and BASIC modules may not be combined into the same program because the new BASIC compiler does not set the bit. Plink can match the common names properly.

Problems have been reported with linking modules produced by Microsoft's new BASIC compiler version 5.3. Plink-I can't link these with the runtime module: the /O switch must be used at compile time. In addition, some special options must be used.

The problem is that the initialization code in these BASIC programs attempts to clear the blank common (.BLNK.) and counts on having the data segment for the module ('BASIC) immediately following .BLNK. in memory, in order to determine its ending address. Plink normally moves .BLNK. to the end of the program so that FORTRAN programs can access free memory; therefore, the clearing routine wipes out the operating system.

To correct the problem, enter
LOCATE .BLNK.=110, 'BASIC=110
at the end of the Plink input command, and do not use blank common in the BASIC program: give each common block a name. Alternatively, locate 'BASIC up high enough to reserve the needed amount of space in .BLNK. The memory map should be checked to make sure that 'BASIC follows .BLNK. in memory before executing the program: anything inadvertently placed between the two will be smashed.

The Microsoft relocatable file format has become an industry standard for Z80 CP/M compiler output. However, several manufacturers are selling compilers which output files that look like Microsoft's but actually contain subtle differences. Plink can handle some of these, but others will not work. Also, Microsoft periodically makes minor changes to the format so as to support new language features.

The following list shows which compilers have been checked out with Plink Version 3.28: 1) Microsoft COBOL 4.01 (in this version the SECTION statement may not be used), FORTRAN 3.31, BASIC 5.3 (see comments above); 2) Cromemco COBOL 3.01; 3) Digital Research PL/I-80 1.3 (the indexed .IRL files are not supported - they have to be converted to normal form with the LIB program); MT Microsystems' Pascal MT (.ERL files designed to be input to the disassembler are not supported); Ithaca Intersystems' Pascal/Z 3.0. If a compiler you wish to use is not on the list, insure that it outputs a format compatible with a listed one, or contact your software distributor.

Plink-II
Version 1.10

Here are the bugs fixed in this new version:

1. Some versions of the PL/I-80 compiler generate invalid .REL files when an empty string is declared. See number 6 above under Plink I.
2. If more than two undefined segments exist (warning #55), Plink-II would die with diagnostic error #164. Now programs may be linked with undefined segments: they are ignored. This is useful for tasks such as the creation of overlay structures that are used in many programs where not all of the segments mentioned in the structure are used in each program.
3. DEFINE <symbol> = <local variable> would cause a syntax error; this has been corrected.
4. Common blocks would not be handled properly if they had the same name as a module: each instance of the common was assigned a different memory area. This situation has been remedied. See number 7 above under Plink-I.
5. The overlay loader no longer incorrectly links in a .COM program when the LOCATE command is used.

A number of additional features are included in the new version. When the LOCATE statement is used in a .PRG program a new section is created. Since only the main section is loaded by the operating system when the program is executed, Plink-II includes the overlay loader into the program to

(continued next page)

load the other section before execution of the program begins. This action may now be inhibited by using the new PAD option provided with the LOCATE command. For example,

LOCATE 3000, PAD

causes the previous section to be filled out until it reaches address 3000. In this way, the next section is forced to be adjacent to the previous one and will be loaded at the same time from disk: the overlay loader is not required.

The COMMON command has been added, and may be used to define the size of a common block to be larger than the definition of the common provided by any module. It also performs the function of the SEG command by placing the common block in the current section. the syntax is COMMON <name 1> = <size1>, <name2> = <size2>, etc. For example,

COMMON C1=100, C2 =#1 * 5

The COMMON command may not be used on a .DATA. segment or error #18 will result. Also, the COMMON and CONCATENATE statements should not be used on the same common or the results will be unpredictable.

The various overlay loaders selected via the DEBUG and I8080 commands are now combined into a single library file called OVERLAY.REL instead of being in separate files. Each has a different module name: .OVLS. is the standard loader; .OVLD. is selected when the DEBUG command is used, and .OVL8. when the I8080 command is used. Older overlay .REL files should be discarded: they will no longer work.

Plink-II also has problems with linking modules produced by Microsoft's new BASIC Compiler. (See the notes on Plink-I above.) If the /O switch is NOT used in the BASIC compile, the program is linked so as to use the runtime support module, and no extra Plink-II commands are required: everything is sorted into the correct order in memory. A .COM file must be specified for output, overlays may not be used, and the LOCATE, ACTUAL, and .DATA. commands should not be used.

If the I/O option IS used, the SEGMENT command of Plink- II can be used to specify that the named segments are to be allocated memory at the current load address. Older versions of Plink-II always put .BLNK. at the end of the current section, but version 1.10 will not do this if .BLNK. is used in the SEGMENT command. Plink-II will move all uninitialized segments to the end of the section, and .BLNK. and 'BASIC are normally uninitialized, so the NOSORT command must also be used to inhibit this action. To sum up, entering

SEGMENT .BLNK., 'BASIC NOSORT

into the Plink-II command will cause the correct memory structure to be generated.

PRISM
Version 2.0.1

Three bugs have been fixed in this update.

1. If a duplicate key was encountered on a key defined as unique while adding a new record, the resulting key contained the correct value while the data record retained the incorrect value. This caused the wrong record to be retrieved when accessed by that key. A fatal error could also occur if an attempt was made to delete the record or change the incorrect key field.
2. Keys defined with numeric fields containing a decimal point were created one character too short; leading digit may have been erroneously truncated.
3. Entry pattern was some times displayed incorrectly for number fields defined with no whole digits (e.g., a field three digits long with three decimal places).

This version has been enhanced by an additional file storage mode, called Extended (as opposed to Normal). The storage mode controls how the data records are constructed physically. PRISM operation is the same in each case.

Normal format is totally compatible with standard CBASIC records, using commas as field delimiters, and quotes (") to denote string fields. In this way, the data file may be read directly with a CBASIC READ statement. Extended format builds data records with no commas or quotes by establishing fixed positions for the fields within each record. This makes all 254 characters of the record available for storing data. An additional advantage is that fields may contain imbedded quotes. Although extended format records are not directly compatible with the standard CBASIC record format, they may still be read with the READ LINE statement. File storage mode may not be altered once data is entered into a file.

New Products

The products described here are available from their authors, software distributors, computer dealers, and software publishers.

baZic
by Micro Mike's, Inc.

This Binary-Coded Decimal BASIC interpreter uses the Z80 instruction set and is designed to be about 40% faster than North Star BASIC(with which it is compatible) or Microsoft's BASIC interpreter. It includes 8, 10, 12 and 14 digit precision, with software and hardware floating point versions.

BaZic is supported by CP/M, North Star DOS, and MicroDoz. It requires a Z80 CPU, 32K RAM space, and a minimum disk capacity of 80K. One drive can be used. A cursor-addressable terminal and CP/M 2.0 or later are also necessary.

Level II COBOL
by Micro Focus Inc.

This implementation of the highest ANSI standards for COBOL includes the following language modules: nucleus, table handling, sequential I/O, relative I/O, indexed I/O, inter-program communication, and sort/merge. Level II COBOL mainframe application programs written to ANSI 74 COBOL standards will be portable to microcomputers with a minimum of conversion.

Level II COBOL is available for 8086 processors equipped with a C compiler.

Dental 80/Medic 80
by The Systems Shoppe

These dental and medical accounting systems includes interactive entry, interactive inquiry, a collection system to remind the user of accounts which are in arrears, patient scheduling, patient registration, and patient history retention. Records can be accessed by patient name or account number. Scheduling exception reports and dental or medical records "pull" lists can also be implemented. Follow-up control is also a design feature. Security measures are intended to restrict information access and to recover files in the event of a power failure.

The following accounting reports are supported: aged A/R report, preauthorized request control and reporting, past due and credit balance accounts, patient statements, demand patient statements, insurance statements including ADA form, demand insurance statements, production by dentist and clinic or group, clinic analysis, patient charges and account balance, recall reminders and letters, referring doctor or dentist reports, daily revenue and production reports, daily audit of all posted transactions with cumulative A/R balance.

Fileshare
by Micro Focus Inc.

Fileshare is an updating environment for MP/M systems running COBOL programs. It is available on a trial basis. Files can be opened for simultaneous updating and different operations can be performed at the same time in a multi-user environment. Each user's files are protected by a mechanism which locks and unlocks records within the files. This is performed automatically by input and output statements in the CIS COBOL program. IBM 8100 protocol has been employed in the development of Fileshare.

Fileshare is supplied as a run time system included in each application program and as a central file server. The server and each application exist in separate MP/M environments. Server and run time system communicate via a packet-based communication protocol. CIS COBOL and MP/M are prerequisites.

FORTH Application Modules
by Timin Engineering Company

This variety package of FORTH source codes provides FORTH definitions not previously published, along with data structures, software development aids, string manipulators, an expanded 32- bit vocabulary, a screen calculator, a typing practice program and a menu generation/selection program. Examples of recursion. <BUILD...DOES> usage, output number formatting, assembler definitions, and conversational programs are supplied. A hundred screens of documentation and a hundred screens of software are on the disk.

The screens may be used with Timin or other FIG FORTH.

Magic Typewriter
by California Digital Engineering

This package is intended to be an all-purpose tool, comprising word processing and database management features. Word processing commands are generally one word or mnemonics. File deletes, renaming, block moves, and search and replace are supported. A line editor is included in the package.

Text formatting features allow odd-even page headings, page numbers, justification, centering, right or left justification, and embedded formatting commands. Files can access other files through embedded commands, or sequential printing of files can be accomplished.

Record lengths in the database are limited to 200 characters and the format is free form, with variable length fields separated by semi-colons. Wild card searches are allowed within the reformatting provisions of the package. Printing can be done selectively, based on the presence or absence of a key string in a specific position.

An expanded version of this product, Magic Typewriter 3, allows full command procedure files to be executed from disk. Merging of two or more files is also permitted, for mailings. A special feature of Magic Typewriter is its ability to automatically handle standard film script format.

New Publications

Experiments in Artificial Intelligence for Small Computers
by John Krutch
This is a practical book presenting programs written in Microsoft Level II BASIC, including game-playing programs and concentrating on problem solving through reasoning. One program stores data and makes deductions using it. A chapter is devoted to natural language processing or verbal communication.

Sams' Microcomputer Dictionary
by Charles J. Sippl
This dictionary includes more than five thousand terms are described, with drawings and photos included. Appendices focus on microprocessor, microcomputer markets, hand-held computers, robotics, etc.

16-Bit Microprocessors
This is a survey-type book by a group of authors, who have compared and evaluated the following microprocessors: the 8086 (Intel), the Z8001 and Z8002 (Advanced Micro Devices and Zilog), the 9900 (Texas Instruments and AMI), the LSI-11 (DEC), the 68000 (Motorola and Rockwell International), and the 16000 (National Semiconductor). Basic concepts are explained and software benchmarks with specifications for comparing processors are provided. Addressing modes, instruction sets, interfacing and software examples are included.

Bugs

dBase II
Version 2.02

You cannot comment out program lines with the asterisk if they contain ampersand variables. dBase will still try to evaluate the line, causing an error. You also cannot put ampersand variables after ENDDO and ENDIF. Usually text on the same line after these keywords is ignored. But when the text contains ampersands, an error is generated.

The record pointer may be lost if anything is done in a loop between a LOCATE and a CONTINUE. Use SKIP instead to advance the record pointer. For example:

(continued next page)

```
locate next 5000 for lname$"A"
store "N"  to notfound
do while .not. eof .and. !(notfound)='N'
     display
     @ 0,0 say "is this it? " get notfound
     continue
enddo
```

will not work properly. Another anomaly is the way DBASE can change the type of a variable.

```
store '                    ' to condition
@ 0,0 say 'Condition ? ' get condition
locate next 5000 for &condition
do while .not. eof
     (do something)
     continue
enddo
```

You will find if you examine the value of '&condition' within the loop that it has been replaced with the boolean value of the condition it represents, i.e., if the match has been found, '&condition' will be equal to '.T.', and if not, it will be equal to '.F.'

If you are using a multi-key index to a single index file, you must change a numeric field into its character string equivalent.

```
I.E.,    rather  than

INDEX ON fieldname+number to
                              indexfilename
       use

     INDEX ON fieldname+STR(number,6) to
                                   indexfilename
```

Microspell
Version 4.3

Invert is intended to allow the LEX file it inverts to reside on any drive in the system, but only will look for the file on the logged in drive.

PAS-3
Version 1.61

The manual states that the DUMP utility file will list the patient data file in alphabetical order. In reality it lists the patient file using ascending patient numbers.

Using The Apple Corvus Module With The Mirror Backup

1. Put the Corvus interface in slot 6.
2. Put the disk controller in slot 4.
3. Run "Bringup s4,d1", the APPLEDOS basic program.
4. The program will ask several questions with obvious answers, and then you will get "BREAK in 205".
5. Type 205<CR> to erase line 205.
6. Type "Run to continue.
7. The program will print garbage on the screen. This is the CP/M directory on the Corvus - no cause for alarm.
8. When you get the prompt back, run the BASIC program "Mirror".

BASCOM-Compiled Files Under CDOS

The Microsoft BASIC compiler run-time module does a test of the operating system to determine which system calls are legal. If the run-time module determines that it is running under CP/M 2.0 or above, system calls 33 and 34 (read random, write random) will be used for random access. Otherwise code will be generated to do this as in CP/M 1.4.

The test is performed via a system call 12. In CP/M 2.x this call returns the version number, always non-zero, in HL. In CP/M 1.x, system call 12 lifts the head the of current drive, and is presumed to always return 0 in HL. But in CDOS, system call 12 (deselect the current drive) apparently returns non-zero in HL.

Programs compiled with BASCOM (such as FPL) may think when they run under CDOS that they are running under a CPM 2.x as a result of this test, and generate system call 33 which in CDOS is not a random read call. This results in confusing error messages like "Can't open file' or "File not found' the first time a read is attempted.

In general, this applies to any program that uses CP/M 2.x random file I/O after testing for version number.

Back Issues

Lifelines begins all new subscriptions with the upcoming issue. However, you can order back issues, as available, at the single copy price. Here is a list of the back issues and their feature articles. (Of course all issues contain our regular features: New Products, New Versions, Bugs, Tips and Techniques.) If you wish, you may use this page as an order form and check the available back issues which you would like. Or give us a call at (212) 722-1700. All orders must be pre-paid, either by check, VISA, or MasterCard. Checks must be in U.S. dollars, drawn on a U.S. bank, and made payable to Lifelines Publishing Corporation. The price for issues sent to the U.S., Canada, or Mexico is $2.50 per back issue. For copies sent to all other countries the price is $3.60. Some copies are available in Xerox copies only. These are slightly higher: single copies are $3.10 and $4.25 for foreign orders. These issues are denoted in italics.

VOLUME I

☐ *JUNE 1980 (Vol. I, #1):*
BASIC Comparisons-CBASIC by Bill Burton
The Undocumented Z80 Opcodes by Robert Halsall
DU Tutorial by Ward Christensen

☐ *JULY 1980 (Vol. I, #2):*
Special features on the CP/M Users Group, including catalogs and abstracts
BASIC Comparisons-BASIC-80 by Bill Burton

☐ *AUGUST 1980 (Vol. I, #3):*
BASIC Comparisons-BASIC- 80, part 2 by Bill Burton
Assembly Language Development Systems by Ward Christensen

☐ *SEPTEMBER 1980 (Vol. I, #4):*
BASIC Comparisons- BASIC-80 Compiler, Rev. 5.2 by Bill Burton
BSTAM-A File Transport Utility by Michael Posehn
The Software Evaluation Group by Steve Patchen
Three Pascals Are Better Than One by Michael Posehn

☐ *OCTOBER 1980 (Vol. I, #5):*
Assembly Language Development Systems, part 2 by Ward Christensen
How To Use a Data Management System by Steve Patchen
A CP/M Patch for Altair and iCOM Systems
The Software Evaluation Group Review Format by Ed Paulette and Steve Patchen

☐ *NOVEMBER 1980 (Vol I, #6):*
A Review of BSTMS from the Publisher
Printing sans LPRINT by Bill Norris
BASIC Comparisons-XYBASIC by Bill Burton
Assembly Language Development Systems part 3 by Ward Christensen
Introduction to Data Management Systems by Timothy Berla and John Lehman

☐ *DECEMBER 1980 (Vol I, #7):*
Business Application Problem Definitions by Steve Patchen and The Software Evaluation Group
Assembly Language Development Systems-SID by Ward Christensen
The CP/M Users Group Volume 46-Catalogue and Abstracts

☐ *JANUARY 1981 (Vol I, #8):*
Assembly Language Development Systems-SID Part 2 by Ward Christensen
A Patch for muMATH Version 2.02
The CP/M Users Group Volumes 44 and 45-Catalogues and Abstracts

☐ FEBRUARY 1981 (Vol I, #9):
A Review of VisiCalc and T/Maker by Steve Patchen
The Osborne Packages-Accounts Payable and Accounts Receivable by Martin McNiff
Assembly Language Development Systems-MACRO-80 and LINK-80 by Ward Christensen
The CP/M Users Group Volume 47- Catalogue and Abstracts

☐ MARCH 1981 (Vol. I, #10):
The Configurable Business System by Ed Paulette
The Osborne Packages-General Ledger by Martin McNiff
BASIC Comparisons-SBASIC part 1 by Bill Burton
The CP/M Users Group Volume 48- Catalogue and Abstracts

☐ APRIL 1981 (Vol. I, #11):
Condor by Ed Paulette and Steve Patchen
BASIC Comparisons-SBASIC part 2 by Bill Burton
A Review of PMATE by Harris Landgarten

☐ MAY 1981 (Vol. I, #12):
A Brief Review of PASM, BUG/uBUG, PLINK, and EDIT by Tom Cochran
Comments on SSSFORTRAN by Trevor Marshall
BASIC Comparisons-SBASIC Version 5.3h Part 3 by Bill Burton
The CP/M Users Group Volume 49 - Catalogue and Abstracts

VOLUME 2

☐ JUNE 1981 (Vol. 2, #1):
A Review of PLINK II by Harris Landgarten
The Software Evaluation Group: SELECTOR IV by Tim Berla and Steve Patchen
The Osborne Packages: Payroll by Martin McNiff
The CP/M Users Group Volume 50 - Catalogue and Abstracts

☐ JULY 1981 (Vol. 2, #2)
MDBS, Part 1 by Harris Landgarten
The CPM USERS Group Volume 51 - Catalogue and Abstracts
A Tutorial on Volume 51 by Ward Christensen

(continued next page)

□ AUGUST 1981 (Vol.2 #3):
Large Memory Management Comes to the Z80 by Bernard R. Wess, Jr.
CP/M Bit Map File Allocation by Kelly Smith
Ashton-Tate's dBASE II by Steve Patchen and Ed Paulette
Random Numbers for Microsoft BASIC by James R. Reinders
The CP/M Users Group Volume 52 - Catalogue and Abstracts

□ SEPTEMBER 1981 (Vol. 2, #4)
Bringing Up UCSD Pascal Version 1.5 or 1.0 From Scratch by Kelly Smith
MDBS Part 2-QRS by Harris Landgarten
Terminal Talk by Steve Patchen
Calling SORT As A Subroutine from MBASIC SORT
8080 Programming Tutorial - Introduction and Terminology - Part 1 by Ward Christensen
Assembly Language Interface to PL/I-80 by Michael J. Karas

□ OCTOBER 1981 (Vol. 2, #5):
8080 Programming Tutorial- Terminology by Ward Christensen
Assembly Language Interface to PL1-80 Part 2 by Michael J. Karas
ABBS, CBBS, FBBS, RBBS, ETC, Maybe you'd like to start one too? by Jim Mills
The Dentist's Office: PAS-3 and Univair by Tom Crites
Better Random Numbers by Bill Burton
The CPMUG Volumes 53 and 54 - Catalogues and Abstracts

T/MAKER II Tips

TIP 1 :

Sometimes one runs out of columns. This can happen when you are keeping constants or have a lot of intermediate results which have to be pulled out at the end of a table. Here is one way to free up columns for other uses:

```
                  Column 1    Column 2      Column 3      Column 4
ex                     999         999           999           999

uc0                    sta         stb
uc1*                     2           3
uc2                      +           +             =
```

Between uc2 and uc19 you can use columns 1 and 2 for any purpose.

```
uc19                   fta         ftb
uc20                     +           +                           =

+         Row 1          5          10            40            15
+         Row 2          1           1             5             2
=         Totals         6          11            45            17
cc
```

This tip courtesy Peter Roizen

TIP 2 :

HOW TO ROUND OFF NUMBERS, using extra "ex" lines

```
                      1977       1978         Increase       Total      Rounded
ex                   99.99      99.99          99.99         99.99       99
ac1                    -          +              =
ac2                    +          +                            =
ac3                    +          +                                      =

+      Item A         9.00      11.11           2.11         20.11       20
+      Item B        11.26      14.14           2.88         25.40       25
=+     Total         20.26      25.25           4.99         45.51       46

ex                    99         99             99           99          99
 =+    Round          20         25              5           46          46
```

COMMENTS:
Adding rounded numbers tends to create an error.
Be careful when you introduce such an error.
This is not the recommended way to round numbers.

You can also round in the standard fashion as follows:

```
ex                   99.99      99.99          99.99        99.99       99.99
zv
+                     3.00       4.00           5.00         7.00        8.00
jc1/                   2          3             3             3           3
jc2                   rnd        rnd           rnd           rnd         rnd
=                     1.50       1.33           1.67         2.33        2.67
```

This tip courtesy Mike Olfe

TIP 3 :

When editing a long table with a sum, average, etc. where you wish to view the end of the table after "COMPUTE", turn on 'Frame Mode' ESC F just below the example line EX and exit the editor with the cursor at the bottom of the table. This will leave the final calculations on the screen after "COMPUTE". The calculation starts at the top of the table because the FRAME Mode is enabled.

Note that it is no longer necessary, as it was in earlier versions of T/MAKER, to position the cursor at the beginning of a file in order to save it. T/MAKER II always saves the whole file, regardless of the position of the cursor when exiting the editor.

This tip courtesy Gerry Sawyer

Renew

If your subscription began in November of 1980 (Volume I, Number 6), time is running out for your subscription renewal. As you can see, we have more and more vital reviews and in store for you. Rather than miss out on any valuable information, send in your subscription form (Lifelines Circulation Dept., 1651 Third Ave., New York, N.Y. 10028) or a letter right away. Or call us at (212) 722-1700 and use your MasterCard or VISA to renew.

Coming Soon

Many more CPMUG disks are in store for you — soon! And we'll be featuring a candid review of the new Osborne I computer. In addition, some comparisons of public domain software and copyrighted software are in the works. Ward Christensen will be continuing his tutorial on 8080 programming and we're expecting more coverage of database management systems.

Some More Software Tricks

by Kelly Smith

Kelly Smith has a couple more "tricks" to go with the one featured on page 36 of our September 1981 issue (Volume I, Number 4). The ORI trick is designed to confuse disassembly and should be well commented in your source code, if you dare to use it. When entering at AND$FUNCTION, the ORI picks up the "XRA A" as F6 Hex, and automatically sets the flags at non-zero.

You might code:

```
and$function:               ;indicate boolean AND
;
          mvi  a,1          ;set flags to non-zero, this is AND
          jmp  do$boolean   ;do boolean function
;
or$function:                ;indicate boolean OR
;
          xra  a            ;set flags to zero, this is OR
;
do$boolean:                 ;boolean functions come here
;
          your code...      ;do something! anything!
          .
          .
          .
```

But consider the following:

```
ori   equ  0f6h equate first byte of ORI n
;
and$function:               ;indicate boolean AND
;
          db   ori          ;set flags with A reg. not zero,
                            ;first byte of ORI
or$function:                ;indicate boolean OR
;
          xra  a            ;set flags to zero, this is OR
;
do$boolean:                 ;boolean function come here
;
          your code         ;is everyone confused?
          .
          .
          .
```

What follows is an example (not fast) for register "swapping" when all registers are used and must be saved.

```
exchange$bc$with$hl:        ;exchange B&C Regs. with H&L Regs.
;
          push b            ;put b&C Regs. on the stack
          xthl              ;H&L Regs.=top stack entry=B&C Regs.
          pop  b            ;B&C Regs.=original H&L Regs.
```

Very often you will code a routine to pass a constant to a subroutine, such as:

```
          mvi  c, 1
          call dumb$subroutine
          .
          .
          .
          mvi  c, 2
          call dumb$subroutine
          .
          .
          .
          mvi  c, 3
          call dumb$subroutine
          .
          .
          .
;
dumb$subroutine:          ;use argument passed in C Reg.
          .
          .
          .
```

By manipulating the return address, you can save one byte per CALL as follows:

```
          call trick$subroutine
          db   1          ;put constant in "return" location
          .
          .
          .
          call trick$subroutine
          db   2          ;put constant in "return" location
          .
          .
          .
          call trick$subroutine
          db   3          ;put constant in "return" location
          .
          .
          .
;
trick$subroutine          ;trick subroutine to get constant
;
          xthl            ;H&L Regs.=return address
          mov  c,m        ;get constant pointed to by H&L Regs.
          inx  H          ;bump for return address
          xthl             ;restore the return address and H&L Regs.
;
dumb$subroutine:          ;use argument passed in the C Reg.
          .
          .
          .
```

(continued next page)

This trick could save you a few bytes, by faking an "indirect jump" via the stack; you might code this routine:

```
         call get$data$word
         jmp  use$data$word
;
get$data$word:                     ;get word into H&L Regs.
;
         lhld my$data$word         ;fetch my data word
         ret
;
use$data$word:                     ;use data word in H&L Regs.
```

But a more elegant (though perhaps obtuse) method could be coded:

```
         lxi  h, use$data$word ;make "indirect access"
         push h                ;save it on the stack
;
get$data$word:                 ;get word into H&L Regs.
;
         lhld my$dat$word      ;fetch my data word
         ret                   ;pop stack for address and "jump"
```

This can lead to even trickier manipulation on the stack for return addresses. At one time or another everyone has coded a routine to "filter" keyboard characters, and it usually looks like this:

```
         cpi  '.'        ;period character?
         jz   filter
         cpi  ','        ;comma character?
         jz   filter
         cpi  ';'        ;semi-colon character?
         jz   filter
         cpi  ':'        ;colon character?
         jz   filter
         .
         .
         .
```

A popular method for "In line" printing of messages in CP/M applications programs is as follows:

```
        lxi  b,filter   ;make "FILTER" address
        push b          ;put "FILTER" address on the stack
        cpi  '.'        ;period character?
        rz              ;pop stack and go, if match
        cpi  ','        ;comma character?
        rz              ;pop stack and go, if match
        cpi  ';'        ;semi-colon character
        rz              ;pop stack and go, if match
        cpi  ':'        ;colon character?
        rz              ;pop stack and go, if match
        pop  b          ;no match, adjust the stack
        .
        .
        .
```

But we need to save some bytes, so we get tricky with coding like this:

```
          call start      ;go to START, after message
          db   'My Junk Program Version 1$'
;
start:    pop  d          ;get address of message string
          mvi  c,9        ;CP/M print string function
          call 5          ;let CP/M do the work
```

The tricky way to move the D&E Regs. to the B&C Regs. might be as follows:

```
          push d
          pop  b
```

But the obvious way, and faster way, is just:

```
          mov  d,b
          mov  e,c
```

A really tricky programmer could use the PUSH/POP method to affect the condition code; this "blows away" even experienced programmers when they encounter it in someone's code. Watch this:

```
          mvi  c,081h     ;the "flags"
          .
          .
          .
          push b          ;use "cunning set-up" to confuse
          .
          .
          .
          pop  psw        ;do it to it!
          .
          .
```

This has the effect of moving the B reg. into the A Reg., and moving the C Reg. into the PSW (flags), with the carry and sign bits SET (sign is minus), and all other flags reset to zeroes. This causes most programmers to mumble for hours.

Operating Systems

These operating systems are available from Lifeboat Associates, except where otherwise mentioned.

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

Hard Disk Modules

Description	Version
Corvus Module	2.1
APPLE-Corvus Module	2.1A
KONAN Phoenix Drive	1.8
Micropolis Microdisk	1.92
Pertec D3000/iCOM 4511	1.6
Tarbell Module	1.5
OSI CD-74 for OSI C3-B	1.2
OSI CD-36 for OSI C3-C'	1.2
SA-100A for OSI C3-D	1.2

LIFEBOAT ASSOCIATES

1651 Third Avenue • New York, N.Y. 10028
Tel: (212) 860-0300 • Telex: 640693 (LBSOFT NYK) • TWX: 710-581-2524 (LBSOFT NYK)

To: **Systems Software Houses**
Subject: **SB-86™**
From: **Anthony R. Gold - President**
Date: **October 15, 1981**

Tony Gold

This is an open invitation to systems houses to benefit from the popularization of **SB-86.**

As you may already know, **SB-86** is our name for the DOS operating system which IBM has chosen for their Personal Computer.

Lifeboat Associates is offering systems houses the opportunity to independently support their O.E.M. clients not only in custom systems software integration but also in the sale of the operating system.

Systems houses which offer software support to *bona fide* independent hardware manufacturers may now earn a 15% commission on the O.E.M. unlimited license for the operating system. That means over **$7,000** in addition to whatever other fees are earned directly by way of engineering and other added value.

SB-86 has been adopted as their 16 bit standard operating system for small business computers by IBM, Microsoft and Lifeboat Associates. Together with others, they will supply some of the hardware, languages and applications software which will ensure that the environment is a *de facto* standard.

Additionally, Lifeboat is eager to discuss with other interested parties the mechanics of making this a *de jure* software standard. The SB in **SB-86** means **Software Bus**™ and it reflects our attitude that the definition and publication of the specification is all important.

And the specification for the standard must adapt with time and new needs. We will encourage others, such as authors considering writing language translators or even compatible operating systems, to participate in further development of the environment's definition. The standard must evolve over time to meet the changing requirements of manufacturers, software houses and users.

As this new generation of hardware and software emerges, we look forward to working with you on this exciting program.

VERSION LIST

October 15, 1981

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

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

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

Product	S	M	OS	P	MR	
ACCESS-80	1.0		CP/M	8080/Z80	54K	
Accounts Payable/Cybernetics	3.1		CP/M	Z80	64K	Needs RM/COBOL
Accounts Payable/MC	1.0		CP/M	8080/Z80	56K	For CP/M 2.2
Accounts Payable/Structured Sys	1.3B		CP/M	8080	52K	w/It Works run time pkg.
Accounts Payable/Peachtree	07-13-80		CP/M	8080	48K	Needs BASIC-80 4.51
Accounting Plus			CP/M	8080/Z80	64K	
Accounts Receivable/Cybernetics	3.1		CP/M	Z80	64K	Needs RM/COBOL
Accounts Receivable/MC	1.0		CP/M	8080/Z80	56K	CP/M 2.2
Accounts Receivable/Peachtree	07-13-80		CP/M	8080	48K	Needs BASIC-80 4.51
Accounts Receivable/Structured Sys	1.4C		CP/M	8080	56K	w/It Works run time pkg.
Address Management System	1.0		CP/M	8080		Requires 2 drives
ALDS TRSDOS		3.40	TRSDOS	8080	32K	TRSDOS Macro-80
ALGOL 60	4.8C		CP/M	8080	24K	
ANALYST	2.0		CP/M	8080	52K	Needs CBASIC2,QSORT/ULTRASORT
APL/V80	3.2		CP/M	Z80	48K	Needs APL terminal
Apartment Management (Cornwall)	1.0	1.0	CP/M	8080		Needs CBASIC2
ASM/XITAN	3.11		CP/M	Z80		
Automated Patient History	1.2		CP/M	8080	48K	
BASIC Compiler	5.3	5.3	CP/M	8080	48K	
BASIC-80 Interpreter	5.21	5.21	CP/M	8080	40K	w/Vers. 4.51,5.21
BASIC Utility Disk	2.0	2.0	CP/M	8080	48K	
BOSS Financial Accounting System	**1.08**		**CP/M**	**8080**	**48K**	**Needs 2/3- drives w/min 200k each, & 132-col. printer**
BOSS Demo	**1.08**		**CP/M**	**8080**	**48K**	
BSTAM Communication System	4.5	4.5	CP/M	8080	32K	
BDS C Compiler	1.44	1.44T	CP/M	8080	32K	w/'C' book
Whitesmiths' C Compiler	2.0		CP/M	8080	60K	
BSTMS	1.2	1.2	CP/M	8080	24K	
BUG / uBUG Debuggers	2.03		CP/M	Z80	24K	
CBASIC2 Compiler	2.08		CP/M	8080	32K	w/CRUN(2,204P, & 238)
CBS Applications Builder	1.3		CP/M	8080	48K	Needs no support language
CIS COBOL Compiler	4.4,1		CP/M	8080	48K	
CIS COBOL Compact	3.46	3.46	CP/M	8080	32K	
FORMS 1 CIS COBOL Form Generator	1.06	1.06	CP/M	8080		
FORMS 2 CIS COBOL Form Generator	1.1,6a	1.16	CP/M	8080		
Interface for Mits Q70 Printer			CP/M			CP/M 1.41 or 2.XX
COBOL-80 Compiler	4.01	4.01	CP/M	8080	48K	
COBOL-80 PLUS M/SORT	4.01		CP/M	8080	48K	
CONDOR	1.10		CP/M	8080	48K	
CREAM (Real Estate Acct'ng)	2.3		CP/M	8080	64K	CBASIC needed
Crosstalk	1.4		CP/M	Z80		
DATASTAR Information Manager	1.101		CP/M	8080	48K	
Datebook	2.03		CP/M	8080	48K	Needs 80x24 terminal
dBASE-II	2.02A		CP/M	8080	48K	
dBASE-II Demo	2.02A		CP/M	8080	48K	
Dental Managememt System 8000	8.7A		CP/M	8080	48K	Needs CBASIC
Dental Management System 9000	**1.06**		**CP/M**	**8080**	**48K**	**Needs CBASIC**
DESPOOL Print Spooler	1.1A		CP/M	8080		
DISILOG Z80 Disassembler	4.0	4.0	CP/M	Z80		Zilog mnemonics
DISTEL Z80/8080 Disassembler	4.0		CP/M	8080/Z80		Intel mnemonics,TDL extensions
EDIT Text Editor	2.06		CP/M	Z80		
EDIT-80 Text Editor	2.02	2.02	CP/M	8080		
ESQ-1	2.1		CP/M	8080		Needs CBASIC2
FABS	2.4A		CP/M	8080	32K	
FILETRAN	1.20		CP/M		32K	1-way TRS-80 Mod I,TRSDOS to Mod I CP/M
FILETRAN	1.4		TRSDOS		32K	2-way TRS-80 Mod I,TRSDOS & Mod I CP/M
FILETRAN	1.5		CP/M		32K	1-way TRS-80 Mod II,TRSDOS to Mod II CP/M
Financial Modeling System	2.0		CP/M		48K	
Floating Point FORTH	2		CP/M	8080/Z80	28K	
Floating Point FORTH	3		CP/M	8080/Z80	28K	
FORTRAN-80 Compiler	3.43	3.43	CP/M	8080	36K	
FORTRAN Package	3.40		TRSDOS			
FPL 56K Vers.	2.51		CP/M	8080	56K	
FPL 48K Vers.	2.51		CP/M	8080	48K	
General Ledger/Cybernetics	1.3C		CP/M	Z80	48K	Needs RM/COBOL
General Ledger/MC	1.0		CP/M	8080/Z80	56K	CPM 2.2 or MPM
General Ledger/Peachtree	07-13-80		CP/M	8080	48K	Needs BASIC-80 4.51
General Ledger/Structured Sys	1.4C		CP/M	8080	52K	w/It Works Package
General Ledger II/CPaids	1.1		CP/M	8080	48K	Needs BASIC-80 4.51
GLECTOR Accounting System	2.02		CP/M	8080	56K	Use w/CBASIC2,Selector III

(continued next page)

VERSION LIST

Product	S	M	OS	P	MR	
GLECTOR IV Accounting System	1.0		CP/M	8080		Needs Selector IV
HDBS	1.05A		CP/M	+	52K	
IBM/CPM	1.1		CP/M	8080		
Insurance Agency System 9000	**1.06**		**CP/M**	**8080**		**Needs CBASIC**
Integrated Acctg Sys/Gen'l Ledger			CP/M	8080	48K	Needed for 3 pkgs. below
Integrated Acctg Sys/Accts Pyble			CP/M	8080	48K	
Integrated Acctg Sys/Accts Rcvble			CP/M	8080	48K	
Integrated Acctg Sys/Payroll			CP/M	8080	48K	
Interchange			CP/M	Z80	32K	
Inventory/MicroConsultants	5.3		CP/M	8080/Z80	56K	Needs CP/M 2.2
Inventory/Peachtree	07-13-80		CP/M	8080	48K	Needs BASIC-80 4.51
Inventory/Structured Sys	1.0C		CP/M	8080	52K	w/It Works Package
Job Cost Control System/MC	1.0		CP/M	8080/Z80	56K	Requires CP/M 2.2
JRT Pascal System	1.4		CP/M	8080	56K	
LETTERIGHT Text Editor	1.1B		CP/M	8080	52K	
LINKER			CP/M	Z80		
MAC	2.0A		CP/M	8080	20K	
MACRO-80 Macro Assembler Package	3.43	3.43	CP/M	8080/Z80		
Magic Typewriter	**3**		**CP/M**	**Z80**	**48K**	
Magic Wand	1.11		CP/M	8080	32K	
MAGSAM III	4.2		CP/M	8080	32K	
MAGSAM IV	1.1		CP/M	8080	32K	Needs CBASIC
MAILING ADDRESS Mail List System	07-13-80		CP/M	8080	48K	
Mail-Merge	3.0		CP/M	8080		
Master Tax	1.0-80		CP/M	8080	48K	
Matchmaker			CP/M	8080	32K	
MDBS	1.05A		CP/M	+	48K	
MDBS-DRS	1.02		CP/M	+	52K	
MDBS-QRS	1.0		CP/M	+	52K	
MDBS-RTL	1.0		CP/M	+	52K	
MDBS-PKG			CP/M	+	52K	w/all above MDBS products
Microspell	4.21		CP/M	8080	48K	Needs 15 K
Medical Management System 8000	8.7a		CP/M	8080		Needs CBASIC
Medical Management System 9000	**1.06**		**CP/M**	**8080**		**Needs CBASIC**
Microcosm			CP/M	Z80		CP/M 2.X or MP/M
Microspell	4.3		CP/M	8080	48K	Needs 150K/drive
Mince	2.6		CP/M	8080	48K	
Mince Demo	2.6		CP/M	8080	48K	
Mini-Warehouse Mngmt. Sys.	5.5		CP/M	8080	48K	Needs CBASIC
Money Maestro		1.1	CP/M	8080/Z80	48K	CP/M 1.4 or 2.2
MP/M-I Operating System	1.1		MP/M	8080	32K	
MP/M-II	**2.0**		**MP/M**	**8080**	**48K**	
MSORT	1.01		CP/M	8080	48K	
Microstat	2.01		CP/M	8080	48K	Needs BASIC-80, 5.03 or later
Mu LISP-80/Mu STAR Compiler	2.10	2.12	CP/M	8080		
Mu SIMP / Mu MATH Package	2.10		CP/M	8080		muMATH 80
NAD Mail List System	3.0D		CP/M	8080	48K	
Nevada COBOL	2.0		CP/M	8080	32K	
Order Entry w/Inventory/Cybernetics			CP/M	Z80		Needs RM/COBOL
Panel	2.2		CP/M		44K	Also for MP/M
PAS-3 Medical	1.77		CP/M	8080	56K	Needs 132-col. printer & CBASIC
PAS-3 Dental	1.63		CP/M	8080	56K	Needs 132-col. printer & CBASIC
PASM Assembler	1.02		CP/M	Z80		
Pascal/M	4.02		CP/M	8080	56K	
PASCAL/MT Compiler	3.2		CP/M	8080	32K	
PASCAL/MT+ w/SPP	5.25		CP/M	8080	52K	Also has SuperBr'n & 32K ver., Needs 200K/drive
PASCAL/Z Compiler	4.0		CP/M	8080	56K	
Payroll/Cybernetics, Inc.			CP/M	Z80		Needs RM/COBOL
Payroll/Peachtree	07-13-81		CP/M	8080	48K	Needs BASIC-80 4.51
Payroll/Structured Sys	1.0E		CP/M	8080	60K	w/It Works run time pkg.
PEARL SD	3.01		CP/M	8080	56K	w/CBASIC2,Ultrasort II
PLAN80 Financial Package	2.0		CP/M	8080	56K	Z80/8080
PL/I-80	1.3		CP/M	8080	48K	
PLINK Linking Loader	3.28		CP/M	Z80	24K	
PLINK-II Linking Loader	1.10A		CP/M	Z80	48K	
PMATE	3.02		CP/M	8080	32K	
PRISM/ADS	**2.0.1**		**CP/M**	**8080**	**56K**	**Needs CBASIC, 2.06 or later & 180K/drive**
PRISM/IMS	**2.0.1**		**CP/M**	**8080**	**56K**	**Needs CBASIC, 2.06 or later & 180K/drive**
PRISM/LMS	**2.0.1**		**CP/M**	**8080**	**56K**	**Needs CBASIC, 2.06 or later & 180K/drive**
POSTMASTER Mail List System	3.4	3.4	CP/M	8080	48K	
Professional Time Acctg	3.11a		CP/M	8080	48K	Needs CBASIC2

VERSION LIST

Product	S	M	OS	P	MR	
Programmer's Apprentice			CP/M	8080/Z80	56K	Needs BASIC-80
Property Management System	07-13-80		CP/M	8080		Needs BASIC-80 4.51
Property Manager	1.0		CP/M	8080	48K	Needs CBASIC
PSORT	2.0		CP/M	8080		
QSORT Sort Program	2.0		CP/M	8080	48K	
Real Estate Acquisition Programs	2.1		CP/M	8080	56K	Needs CBASIC
Remote	3.01		CP/M	Z80		
Residential Prop. Mngemt. Sys.	1.0		CP/M	Z80	48K	
RM/COBOL Compiler	1.3C		CP/M	8080	48K	w/Cybernetics CP/M 2
RAID	4.7.3A	4.7.3	CP/M	8080	28K	
RAID w/FPP	4.7.3A	4.7.3	CP/M	8080	40K	
RECLAIM Disk Verification Program	2.1		CP/M	8080	16K	
SBASIC	5.4		CP/M	8080	48K	
Scribble	1.3		CP/M	8080		
SELECTOR-III-C2 Data Manager	3.24	3.24	CP/M	8080	48K	Needs CBASIC
SELECTOR-IV	2.14A		CP/M	8080	52K	Needs CBASIC
Shortax	1.2		CP/M	Z80	48K	TRSDOS,MDOS too, needs BASIC-80 5.0
SID Symbolic Debugger	1.4		CP/M	8080		N/A-Superbr'n
SMAL/80 Programming System	3.0		CP/M	8080		For CP/M 1.x
Spellguard	**2.0**		**CP/M**	**8080/Z80**	**32K**	**Needs Word Processing Program**
Standard Tax	1.0		CP/M	8080	48K	Needs BASIC-80 4.51
STATPAK	1.2	1.2	CP/M	8080		NeedsBASIC-80 4.2 or above
STIFF UPPER LISP	**2.2**		**CP/M**	**8080**	**48K**	
STRING BIT FORTRAN Routines	1.02	1.02	CP/M	8080		
STRING/80 bit FORTRAN Routines	1.22		CP/M	8080		
STRING/80 bit Source	1.22		CP/M	8080		
SUPER SORT I Sort Package	1.5		CP/M	8080		Max. record=4096 bytes
T/MAKER II	2.3.2		CP/M	8080	48K	Avail. for CDOS
T/MAKER II DEMO	2.2.1		CP/M	8080	48K	
TEX Text Formatter	2.1		CP/M	8080	36K	
TEXTWRITER-III	3.6	3.6	CP/M	8080	32K	
TINY C Interpreter	800102C		CP/M	8080		
TINY C-II Compiler	800201		CP/M	8080		
TRS-80 Customization Disk	1.3		CP/M	8080		
ULTRASORT II	4.1A		CP/M	8080	48K	
Lifeboat Unlock	1.3		CP/M	8080		Use w/BASIC-80 5.2 or above
VISAM	2.1		CP/M	8080	48K	
Wiremaster			CP/M	Z80		Needs 180K/drive
Wordindex	3.0		CP/M	8080	48K	Needs WordStar
Wordmaster	1.07A		CP/M	8080	40K	
WordStar	3.0		CP/M	8080	48K	
WordStar w/MailMerge	3.0		CP/M	8080	48K	
WordStar Customization Notes	3.0		CP/M	8080		
XASM-05 Cross Assembler	1.04		CP/M	8080	48K	
XASM-09 Cross Assembler	1.05		CP/M	8080	48K	
XASM-51 Cross Assembler	1.07		CP/M	8080	48K	
XASM-F8 Cross Assembler	1.03		CP/M	8080	48K	
XASM-400 Cross Assembler	1.02		CP/M	8080	48K	
XASM-18 Cross Assembler	1.30		CP/M	8080		
XASM-48 Cross Assembler	1.30		CP/M	8080		
XASM-65 Cross Assembler	1.95		CP/M	8080		
XASM-68 Cross Assembler	1.96		CP/M	8080		
XMACRO-86 Cross Assembler	3.40		CP/M	8080		
XYBASIC Extended Interpreter	2.11		CP/M	8080		
XYBASIC Extended Disk Interpreter	2.11		CP/M	8080		
XYBASIC Extended Compiler	2.0		CP/M	8080		
XYBASIC Extended Romable	2.1		CP/M	8080		
XYBASIC Integer Interpreter	1.7		CP/M	8080		
XYBASIC Integer Compiler	2.0		CP/M	8080		
XYBASIC Integer Romable	1.7		CP/M	8080		
ZAP-80	**1.4**		**CP/M**	**8080**		**Needs 50K/drive**
Z80 Development Package	3.5		CP/M	Z80		N/A-Magnolia,Superbr'n,mod.CP/M
ZDM/ZDMZ Debugger	1.2/2.0		CP/M	Z80		For N'Star,Apple,IBM 8"
ZDMZ Debugger	2.0		CP/M	Z80		See note above
ZDT Z80 Debugger	1.41	1.41	CP/M	Z80		N/A-Superbr'n,mod.CP/M
ZSID Z80 Debugger	1.4A		CP/M	Z80		N/A-Superbr'n,mod.CP/M

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

If you can't find the right program in our new catalog it probably hasn't been written.

As the world's leading publisher of professional software for microcomputers, Lifeboat Associates offers the largest selection of state-of the art programs. And our new catalog has more to offer than ever, including:

TELECOMMUNICATIONS such as:
RBTE-80™ (REMOTE BATCH TERMINAL EMULATOR)

Allows Z80™ microcomputers to emulate IBM 2770, IBM 2780, IBM 2968, IBM 3741, or IBM 3780 remote batch terminal. Enables transfer of data files to and from mainframe computers or other remote batch terminals at speeds up to 19.2 kilobaud. Provides data integrity using IBM binary synchronous protocol for error detection and retransmission. Advanced features include full IBM Bisynchronous Protocol, hardware diagnostic, dynamic terminal configuration, online communication trace, unattended or attended operation, user customization capability, and more. Also requires Z80 CPU.

LANGUAGES such as:
baZic II

Designed for speed. Takes full advantage of Z80 instruction set. Executes programs written in North Star BASIC on virtually any Z80 based micro, with few or no modifications. Adds functions to aid screen formatting and simplify programming, such as APPEND, ON GOSUB, cursor addressable print, etc. Includes precisions of 8, 10, 12, and 14 digits. Also requires Z80 CPU, CP/M-80® 2.X or later, 32K RAM, and at least 80K disk storage capacity. Cursor addressable terminal desirable for some functions.

KBASIC

Microsoft Disk Extended BASIC version 4.51 integrated with KISS Multi-keyed Index Sequential and Direct Access file management as 9 additional BASIC commands. KISS included as relocatable modules linkable to FORTRAN-80, COBOL-80, and BASIC Compiler. Specify: CP/M 1.4 or 2.x. Available also to licensed users of Microsoft BASIC-80.

WORD PROCESSING SYSTEMS AND AIDS such as:
Spellguard

Spelling proofreader assists in eliminating spelling and typographical errors. Each word in your document is compared with words in its 20,000 word dictionary and all mismatched words are presented as possible errors. You may then correct the word, and may, simultaneously, add a new word to the expandable dictionary. Flags spelling errors with amazing speed and leaves the correcting to you. Valuable for proofreading large volumes of text quickly.

PROFESSIONAL AND OFFICE AID such as:
AMERICAN SOFTWARE PROPERT MANAGEMENT PROGRAM

Comprehensive tool for real estate operators. Designed to store names, addresses, phone numbers, security deposits, scheduled and actually received rents in all properties, whether apartment or office buildings, shopping centers, etc. Permits entry of expenses paid in up to 99 different categories for up to 150 separate properties or more. *Writes* expense checks and check register and actually balances company checking account. Reports available include Operating Statements, Vacancy Report, Late Rent, and more. Also requires CBASIC-2™, 132 column printer, and 24 X 80 CRT with cursor addressing.

Prices reflect distribution on 8" single-sided, single-density diskettes.
The sale of each proprietary software package conveys a license for use on one system only.
All software products have specific requirements in terms of hardware or software support, such as MPU type, memory size, support operating system or language. We recommend the following MINIMUM SYSTEM REQUIREMENTS: an 8080/8085/Z80 system with SB-80™, or other CP/M-80 compatible operating system, 48K bytes of memory, and two floppy disks each with at least 70K bytes of storage.

SPECIAL 50% off CP/M 2.2 configured for Heath H89 by Magnolia. Limitec quantities available. Regularly $250.00. Now $125

MEDIA FORMAT CODES—Lifeboat Associates currently supports over 110 computers. The newest are:

- IBM Personal Computer
- DEC VT18X
- DTC Micro 210A
- EXO
- Hewlett-Packard 125
- Ibex 7000
- Intersil Development System
- Lexitron VT1303
- Micro Mega 85
- Osborne-1
- Xerox 820
- Wang WangWriter
- Zilog MC22-20,25,50

If your computer is not listed above, look for it among the 110 computers listed in Lifeboat's Software With Full Support Catalog.

Write, TWX, Telex or call for more information about these and all of our 200 programs, including:

- System Tools
- Language and Application Tools
- Data Management Systems
- Financial Accounting Packages
- Numerical Problem-Solving Tools
- General Purpose Applications
- Periodicals
- Books
- Accessories

Lifeboat Associates
1651 Third Avenue
New York, N.Y. 10028
Tel.: (212) 860-0300
TWX:710-581-2524(LBSOFT NYK)
Telex:640693(LBSOFT NYK)

Lifeboat, Inc.
OK Bldg., 5F
1-2-8 Shiba Daimon
Minato-ku,Tokyo,105 Japan
Tel.: 03-437-3901
Telex:2423296(LBJTYOJ)

Lifeboat Associates, Ltd.
PO Box 125
London WC2H 9LU, U.K.
Tel.: 01-836-6921
Telex:893709(LBSOFTG)

Lifeboat Associates GmbH
Hinterbergstrasse 9
Postfach 251
CH 6330 Cham, Switzerland
Tel.:042/36 8686
Telex: 865265 (MICO CH)

Intersoft GmbH
Schlossgartenweg 5
D-8045 Ismaning, W. Germany
Tel.: 089-966-444
Telex:5213643(ISOFD)

Lifeboat Associates SARL
10,Grande r. C. de Gaulle
92600 Asnieres, France
Tel.: 01-836-6921
Telex:250303
(PUBLIC X PARIS)

Program names and computer names are generally trademarks or service marks of the manufacturing company. The following are trademarked by the authors or companies shown:
SB-80 and RBTE-80 are trademarks of Lifeboat Associates.
CBASIC-2 is a trademark of Compiler Systems.
CP/M-80 is a registered trademark of Digital Research, Inc.
Z80 is a trademark of Zilog, Inc.

#1
Lifeboat Associates
Software With Full Support

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

Entered as Second Class Matter
at the Post Office, New York, N.Y.