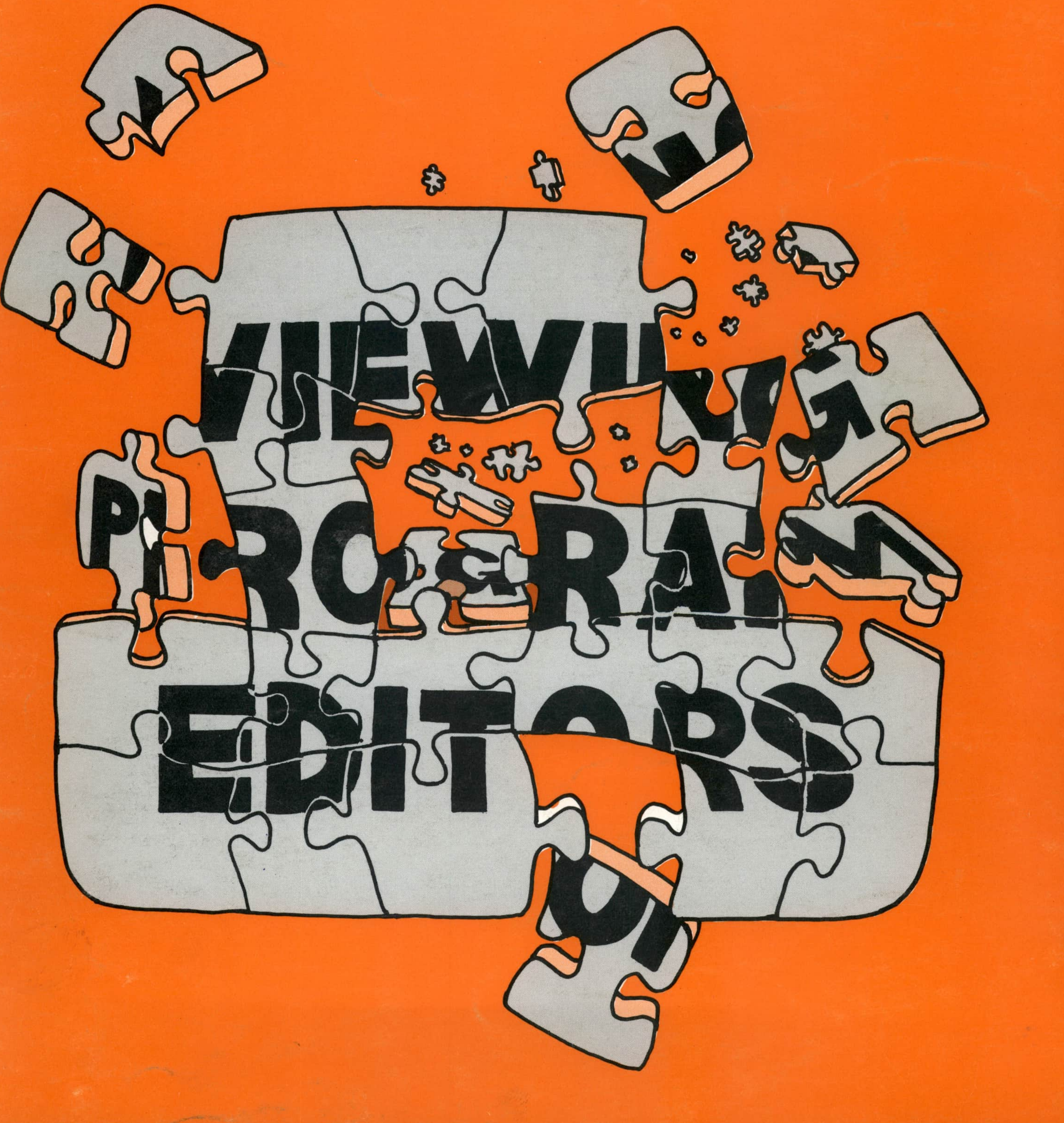
LIFELINES®

THE SOFTWARE MAGAZINE™

$2.50 February 1982 Volume II, No. 9 (ISSN 0279-2575, USPS 597-830)



LIFELINES®

THE SOFTWARE MAGAZINE

February 1982 Volume II, No. 9

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

Contents

Opinion

Features

The single issue price is $2.50 for copies sent to destinations in the U.S., Canada, or Mexico. The single issue price for copies sent to all other countries is $3.60. All checks should be made payable to Lifelines Publishing Corporation. Foreign checks must be in U.S. dollars, drawn on a U.S. bank; checks, money orders, VISA, and MasterCard are acceptable. All orders must be pre-paid. Please send all correspondence to the Publisher at the below address.

Software Notes

Product Status Reports

Miscellaneous

Lifelines (ISSN 0279-2575, USPS 597-830) is published monthly at a subscription price of $18 for twelve issues, when destined for the U.S., Canada, or Mexico, $40 when destined for any other country. Second-class postage paid at New York, New York. POSTMASTER, please send changes of address to Lifelines Publishing Corporation, 1651 Third Ave., New York, N.Y. 10028.

Editorial Comments

THE OPERATING SYSTEM IS THE THING WHEREIN ...

A number of you have requested details on the IBM Personal Computer operating system, as a means of forming your own comparison with other sixteen bit operating systems. Therefore this month a cursory examination of Lifeboat Associates' SB-86 (a.k.a. MSDOS, a product of Microsoft) is provided.

Aside from the fact that all of Microsoft's languages are available for SB-86, i.e. BASIC Interpreter, BASIC Compiler, FORTRAN, COBOL and Pascal, a wide variety of utilities permit conversion of 8080/Z80 assembly language code to 8086 assembly language, reading of single density CPM-80 disks, assembler, linker, etc.

Since SB-86 emulates system calls to CP/M-80 one has merely to run 8080 source code programs through the INTEL conversion program and assemble them with the SB-86 assembler. Almost all 8080 source code will work following conversion to 8086 without further modification. Thus conversion to SB-86 is typically much easier than to any other sixteen bit operating system.

As in the case of UNIX, SB-86 simplifies I/O to various physical devices by using a single set of I/O calls which treat all devices the same from the user's point of view. Therefore, there is no need to rewrite a program when a new physical device is added to a system. One immediate benefit is that such device-independent I/O causes all control characters to be handled by different devices in the same way (e.g. TAB).

Many operating systems exhibit a characteristic rudeness when an error occurs. How many times have you found yourself staring at a screen in a state of frustration trying to decide what BDOS SELECT ERROR means? You soon learned that this catch-all epigram means, among other things, that you are in deep yogurt as far as your file is concerned! The only road out of this swamp is typically a cold boot, or, as we say in the trade, the proverbial "reset to zero!" Thus when such systems blow up while you are editing a file ... well, you know the story ...

When an error occurs SB-86 retries three times the operation which led to the error. If this proves unsuccessful, SB-86 returns an error message and waits for a user response. That's right folks, gone are the days of warping immediately, irrevocably, and irreparably at the speed of light into the twilight zone at the first sign of difficulty. The user can attempt recovery rather than having to reboot the system.

Since SB-86 is a truly relocatable operating system, the relocatable linking loader can provide for separate segments. The COMMAND program in SB-86 relocates the modules during loading rather than loading them to preset addresses. Therefore SB-86 does not have program space and location restricted to the first 64K.

SB-86 has no practical limitation on file or disk size since it uses 4-byte logical pointers compatible with those of Microsoft's XENIX operating system to provide disk and file sizes up to one gigabyte.

Also files of different logical record lengths can reside on the same diskette. A unique feature of SB-86's architecture is built-in blocking and deblocking of physical sectors. Those of you who implement operating systems will appreciate the savings this offers in developing a BIOS. Or, said another way, 128 is no longer a sacrosanct number with of the attendent headaches we all have suffered from.

SB-86 employs the concept of a template for the command line interpreter. The term template refers to the last input line entered. This technique allows editing of a new line, if it should be desirable to re-enter a line, edit it and then execute the line. Text entered at the command line level is placed in an input buffer until the line is terminated by a carriage return. The depression of the return key causes the buffer to be sent to the requesting program for processing. But this same key depression also causes a "template" to be created in the input buffer.

The COMMAND program provided in SB-86 handles communication between the user and the file manager. It allows the user to display directories, rename, delete and copy files, etc. A batch facility is supported which can be entered automatically when the system is booted. In the event that a batch file is not found upon booting, the system immediately prompts the user for the date and time. This is used to date/time stamp files so that you can easily tell the last time a file was modified. Time/date stamping of files allows one to tell whether an arbitrary file is the latest version.

One interesting feature is the PAUSE command. This is invoked in batch files and may be used to prompt the user. This command also suspends all further activity until a carriage return is entered.

EDLIN is a line text editor provided with SB-86 that has lots of bells and whistles. It allows lines of up to 255 characters.

DEBUG is a resident debugger which allows the user to alter a program in memory while debugging. If a command line contains an error it is reprinted by DEBUG and an up-arrow points to the error. Note that errors which occur don't terminate the DEBUG mode but rather the command line entered under DEBUG. One nice feature is the ability to write a file from memory to disk without leaving DEBUG. The SEARCH command allows searching over a specified range of RAM for a list of one or more bytes. The TRACE command uses the hardware trace feature of the 8088/8086 so that it is possible to trace execution of instructions in ROM. Inline disassembly is also supported. Also INPUT and

(continued on page 48)

The Pipeline

Carl Warren

SOME THOUGHTS ABOUT DISK DRIVES

A key element for any microcomputer is the storage system. As a result of the latest innovations in technology, you can find just about anything you desire from floppies to rigid disk drives.

Furthermore, controllers that make the higher performance drives work in a variety of systems are finally coming available.

Making the choice of what to integrate either from an end user's or a systems integrator's point of view can be tough. It can be easier, however, if you have some inkling of what's available and the market dynamics.

Some market dynamics

It may appear obvious that the disk market is rapidly expanding. However, understanding where it's going from a product viewpoint can be something else again. Take for example the playoff of 8-in. low-capacity Winchester compared to the newer 5.25-in. models.

Should everything stay equal, with no changes in capacity or other performance factors, many believe that the market would shape up, by '85, with 8-in. Winchesters stabilizing, and the 5.25-in. soaring on. Although this market projection is based on blue sky, it may not be far from wrong, for several reasons.

The 8-in. market was slow in developing due to difficulties encountered by many manufacturers in solving engineering problems, and slowness in deliveries. Moreover, the real end user (that's the guy who puts the total system to work in his business) wasn't ready to accept the new technology in the volumes expected. This was probably due to the high cost of money, and many unanswered questions such as backup, and what to do when the disk gets full, or just plain breaks.

Furthermore, controller manufacturers were faced with the problem of building controllers without full definition of what the interface was going to be. Consequently, many opted to support the IMI and Shugart drives, thus giving them the market, with Shugart garnering the lion's share (approximately 25,000 units).

Further cluttering the marketing picture was the introduction of the 5.25-in. Winchester at the 1980 NCC in Anaheim. This drive not only excited many, but caused numerous OEM designers to reevaluate previous plans to incorporate the larger 8-in.

(continued next page)

drive. Making the smaller drive even more enticing was the use of a Shugart like interface which meant that existing controller designs could work handily with the small unit.

The picture becomes even more blurred because 8-in. manufacturers are aiming for the low-end of the market that heretofore has been the province of 14-in drives. At the same time, 5.25-in. makers are projecting designs with 60 to 120 Mbytes of storage in as little as two years.

This means of course that the market will resegment. The 8-in. drives will increase in capacity, probably leveling out at 500M bytes; the 5.25-in. market will most likely settle in at about 120 to 200 Mbytes, and the 14-in. market will reach into the very high end, providing storage capacity in the gigabyte range.

Essentially what appears to be happening is that the low-end is being raised. Within a short two years, low-end will not necessarily be equitable to small.

Small capacities mean floppy

Even though the high-capacity range, 10M bytes and above, will be filled by rigid disk technology, floppies will fill the gap on the other side.

Already manufacturers, such as Iomega, and Persci are offering high capacity floppies to fill both backup requirements and serve low-end storage needs. Currently, drives offering 1Mbyte storage are the rule rather than the exception, and 2 and 3M byte versions aren't far behind, with a reported showing by IBM of the so-called Bright project in Japan this past year—a 3Mbyte 8-in drive.

Both Jim Porter, president of Disk Trends, and Ray Freeman, president of Freeman Assoc. agree that the 8-in. IBM compatible floppy have a long life span. Moreover, Porter contends that the newer thinline 8-in. drives will make a significant impact on systems designs. He points out, that the drives most likely won't be used to replace existing units, but will be employed in designs just now on the drawing boards. Porter apparently is right on his projections. Already Tandon has garnered a significant share of the 8-in. market with its thin-line series. But there is a threat on the horizon from Epson America. Reports are that Epson will be introducing high-capacity (1Mbyte/drive) 5.25-in. very thin floppies at NCC in Houston this year.

Porter expects a bullish marketplace for all drive sizes. He however, hasn't put a a handle on smaller floppies like Sony's 3.5-in. model. Freeman, however, is willing to speculate. He believes that the 3.5-in. drive is likely to become a major product class especially in the office machine environment. Moreover, he expects the tiny drive to increase in capacity, and evolve into a rigid drive with an initial capacity of 5Mbytes. Furthermore, Freeman believes that designers can expect this high-capacity mini-mini-rigid rigid by 1983.

But Freeman may be off by a few months on the 3.5-in Winchester. According to industry sources, you can probably expect the baby Winchester to start showing up in volume as early as mid-year with a number of products integrating the product for a full showing at NCC.

Although not confirmed, you might do well to stick around the Sony, Monroe, Osborne, and Otrona booths to get a quick gander at the fledgling drive.

The bottom line

Apparently, the real key to success in cashing in on the disk market, appears to be in guessing the storage needs as far out as 5 yrs. while at the same time providing the right product for now. Many walk this tightrope by grabbing on to any new drive that seems hot at the time, and hoping it can be quickly integrated into a system before the competition.

This method, practiced by a number of system integrators, is cause for alarm, suggests Anova Corp's CEO, Dale Williams. He believes that it's the responsibility of the the system designer to become a problem solver for the end user, rather than the other way around; the designer should try to provide a system that is totally integrated and doesn't require any understanding on the part of the user. He believes that this approach may cause some slowness in the developing market; but in the end will produce a much stronger one.

Others disagree, though, suggesting that regardless of how the end-user market is approached, the disk market for all segments will continue to grow at a rapid pace.

Some hot introductions

As examples of what's come available in the last few months of 1981, consider these drives form Shugart and MPI.

The Shugart Assoc. SA-1100 series of drives offer up to 33.9Mbytes of unformatted capacity, and an average access time of 35msec. Plus Z-8 microprocessor control of essential drive functions.

The high-performance drive, which is available in two models: SA1104 (20.3Mbytes), and SA1106 (33.9-Mbytes) consists of: read/write and control electronics, integrated direct drive brushless DC spindle motor, air filtration system, closed loop servo system, and a shipping/parking lock mechanism.

The drive is designed to be form compatible to the manufacturers SA1000 in size (fits standard 8-in. floppy form factor [4.62-X8.55-X14.25-in.]), and track capacity (500 tpi), and transfer rate (4.34Mbits/sec). Moreover, both the SA1104 and SA1106 employ an SA1000 interface so that existing controller designs can be employed, thus reducing the amount of difficulty in integrating the drives in the system. The only requirement is to add an additional head select to the interface and controller lines.

The SA1100 drives employ the so-called Fastrak closed loop servo positioning system. This system is made up of the read/write heads mounted on a ball bearing-supported carriage which is positioned by a rotary voice coil motor. The bottom surface of the lowest disk contains the continuous servo data used by the closed loop system. This configuration permits the 500 tpi track density and the 35msec average access time.

The voice coil rotary actuator coupled with the track following servo permits precise positioning of the read/write heads on the tracks. This is accomplished by the use of an Automatic Gain Control (AGC) amplifier that amplifies the low level differential signal from the servo pre-amplifier. The Position demodulator provides the position error signal and AGC control signal, which is all coupled with information from the electronic tachometer that tells the actuator its travelling velocity, and the curve generator which sets up the velocity trajectory during a seek. To ensure absolute accuracy, position compensation circuitry provides a positive phase shift for servo control stability. All this information is then fed into a summing amplifier that generates the servo error signal to control the servo power driver.

Other characteristics of the drives are: the SA1104, 20.3Mbyte model employs two platters providing three data surfaces, and one servo surface and uses four read/write heads. The SA1106, on the other hand, uses three platters providing five data surfaces, and one servo surface with six heads.

Both models employ a direct drive brushless DC motor which operates at 3125 RPM, plus eliminates the necessity of providing AC line voltages. Working in concert with the motor is an electromagnetic spindle brake that under control of the integrated uP provides a controlled slowing of the platter during powerdown or power outage situations. The drive requires only +24VDC +/− 10% at 4.0A and +5VDC +/− 5% at 4.0A for the motor and associated logic circuits. In addition, both units provide 6.77 Mbytes (unformatted) capacity per surface (10.4Kbytes per track), at a recording density of 6006bpi.

The integrated Z-8 microprocessor provides intelligent control of the servo system, plus self-testing of the read/write channels, motor control and breaking. And it controls the safety features by monitoring both the drive power and motor speed, bringing the head assembly into a safe area when either begin to fall below the specified operating ranges.

Because rigid media can and do have defects that are a result of manufacturing, the SA1100 drives implement a media defect map on cylinder 659 recorded in the manufacturer's SA1000 format. Defect location information is identified by the byte position with respect to the index mark of the defective track. Moreover, the defective track is further identified by the cylinder and head numbers; its size is specified in length which covers an area of bit cells on the specified track. The defect record information is written in sequence, starting with the smallest cylinder address and increasing order in cylinder number. Shugart provides a hardcopy map of the defect information to assist designers in assigning alternate tracks, or in providing lengthened interrecord gaps (the gap appears over the defect).

You can expect to pay from $1600 to $1900 for these drives in OEM quantities, and systems will more than likely be in the $4000 + range.

For those of you looking for another 5.25-in. Winchester, you might want to at least look at MPI's spec sheet.

The MPI Model 10 Super-Micro Winchester features 10 Mbytes formatted capacity (12.06 unformatted), and an average access time of 25msec which is approximately three times faster than similar sized Winchesters.

The Model 10, which measures 3.25 x 5.75 x 8-in., fits the same form factor as conventional 5.25-in. flexible drives, employs two platters for a total of four surfaces, with 3 Mbytes unformatted capacity per surface (8900 bytes per surface). The formatted capacity is 2.5Mbytes per surface, 7433 bytes per track and 232 bytes per sector, with 32 sectors per track.

The drive which has 371 tracks per inch (tpi),(337.5 cylinders per surface) and a bit density of 8000 bpi, sports a §mbit/sec transfer rate, a track to track access time of 3 msec, and an average latency of 7.5msec.

Interfacing to the host controller is accomplished using either a Seagate ST-506 or Shugart Associates SA 1000 compatible interface.

Power requirements for the model 10 are: +12V at 1.8-3.3A and +5V at 1.5A; typical rotational speed is 4000 RPM, with an ambient operating temperature range of 50 to 115 degrees F. This drive is priced at about $995(500) and will probably top out in a system for under $2000. Of course one has to assume that deliveries will be on time and that the reliability factor will stand up.

Basics of magnetic-head design

Disk drives use magnetic heads to read and write data; like many other things associated with the computer industry there is frequently some misunderstanding as to how they work.

Magnetic heads are nothing more than small high-performance electromagnets. And in operation, they produce a flux at the read/write gap–a tiny opening separating the poles. When 'flux' is produced, it isn't confined to the space between the gap but 'fringes' to either side, thus causing a splattering effect. When this occurs on a diskette, the result can be unwanted information in the track gaps causing errors on read back.

Essentially, there are three ways around this problem. One solution is to provide sufficient gap width between tracks so that fringing would have no effect. This would work but could, for example, reduce the effective tpi of a 48-tpi drive to half. Consequently, this approach is not cost-performance effective. One can also employ either a straddle erase head, or a tunnel erase head.

With the straddle erase head, either side of the read/write core has an erase core whose gap is perpendicular to that of the main gap. This, as a consequence, means that the erase gaps are small, and permits the placing of tracks closer together. When this type of head is writing, the erase heads are turned on and 'straddle' the desired track. This forces the write current to be narrow and follow the center-line of the track. Of course should the head be skewed by more than 9-degrees (the amount IBM says is permissible) the data track will be overwritten by the erase heads.

The tunnel erase head, although performing the same job as the straddle erase, performs it in a different manner. This head, developed and used by IBM, uses two erase cores mounted well behind the read/write gap and core. The direction of flux is parallel to that of the read/write gap, and as a consequence can cause a fringing effect to adjacent tracks. When this type of head is used, it is turned on after the

(continued next page)

read/write head makes a flux reversal. Although this is one way of implementing the head, it can be operated in much the same manner as the straddle erase.

Interestingly, a medium written on drives of similar capacity, but employing different head designs is interchangeable. According to various designers the interchange problem occurs primarily with high density designs.

About thin-film heads

Because information needs are growing the necessity of providing better storage systems is keeping pace. One way disk manufacturers are attacking the problem is by increasing the areal densities of drives.

Designers have concluded that ferrite heads won't permit densities of 1000 tpi and 14,000 bpi (the theoretical limit) simply because the gap can't be machined fine enough, nor the slider reduced to permit lower flying heights. Therefore, many are considering thin-film heads.

By employing semi-conductor methods to manufacture the heads on ceramic substrates, gaps sizes well under 1-uin. can be achieved as can sliders that will fly at 10 to 12-uin.

Many observers expect heads of this class to be used exclusively in high-performance drives in the next two years, or even sooner if yields can be brought up and prices down.

Interestingly, Seagate Technology reevaluated the use of thin-film heads in the model ST-512, and have redesigned the product to use ferrite technology. The problem appears to be that manufacturers of thin-film heads aren't able to deliver reliable heads in volume.

Understand those terms

Another problem that seems to creep into any discussion of disk drives is an inexact understanding of what certain terms mean.

Frequently, designers and marketers let the terms roll off their tongues with little thought as to the accuracy of their meaning. The following glossary, however, should set you straight and maybe give you some extra insight you may not have had before.

Access time–The speed with which a particular sector is found for the writing or reading of data is gauged by the 'access time'. First the head must be positioned over the proper track, which requires a 'seek time', the proper sector of the track must come under the head which involves the 'latency time'. Typical access times range from 10 to 100 msec.

Areal Density–Is the product of the number of bits per unit length (bpi), along the track and the number of tracks per unit length (tpi).

Band actuation–Is a positioning mechanism that permits quick movement of a read/write head assembly to the correct track. Two types of bands exist; the taut band, and the continuous band. The former involves a capstan around which an anchored metal band is wrapped. Movement of the head carriage assembly is controlled by rotating the capstan. In the latter or continuous band method, an idler roller and pulley are used with the carriage attached to the band. The stepper motor moves the band in small steps and the attached carriage moves a calculated distance. Typically, the maximum distance of travel is approximately 0.8-in.

Bit shift–When a data bit is written, it is positioned in the exact center of the bit cell, with either the leading or trailing edge being a clock pulse depending on the encoding scheme. Because the encoding schemes vary to accommodate higher densities, the bit cells become smaller as the capacity gets larger. This means that the bits are closer, crowded together. When read, the head encounters a change in flux and a current is developed in the core. Unfortunately, the current change is not instantaneous and a finite amount of time (typically 350 ns) passes before a peak occurs. While this is going on, the medium is moving placing another bit cell under the head, and although the first transition peaked, it hasn't returned to zero, now the next transition comes along and is summed with the first giving a new peak which is shifted from its proper position. Luckily, these shifts are predictable and can be compensated for.

Data Separation–Data coming from the disk to the controller, is a composite signal composed of both clock and data bits. It is the job of the data separator to pick the data from the clock pulses. In single density floppies, the separation is fairly easy since the data bit is surrounded by clock bits which window the data. In double density the separation isn't as easy, due to the lack of a constant clock. In a case such as this, the data separator must first determine the nominal position of the clock and data bits then generate a 1-usec clock and data window around the bit positions. Data separators are usually analog circuits with a phase-locked loop to provide the necessary time domain resolution needed for high densities.

Droop–This is the difference of signal as the read/write head moves from the outer to inner tracks on a floppy disk. Frequently, at the crossover point, the signal difference is great enough to cause a change–droop–in the generated sine wave. This would, in a system employing a peak detector, cause the detection of an erroneous peak, and thus an error.

Flexure–A magnetic read/write head must be mounted in a manner that makes either contact with medium or permits accurate flying heights to be achieved. With floppies, the problem is easily solved since the head(s) can be mounted on a rigid arm that will apply a given amount of force. However, with Winchester–flying head technology, it becomes important that the head have some freedom in order to 'fly'. To achieve this, the head is mounted in a metal holder called a flexure. The name derives from the fact that the metal does in fact flex–move–with the characteristics of the drive.

Latency–Because a disk is circular, with each track divided into sectors, and because the read/write head is stationary relative to any sector, and because it takes a certain finite amount of time for a sector to come under the head a latency occurs–or time simply passes before the correct sector is located.

Lead-screw–Is a positioner technology, whereby a carefully machined screw-cam, attached to a stepper motor, is employed to move the head in several small steps. This method isn't very fast in comparison to a band design, but is used for accuracy and for drives where interchangeability is an important factor.

Pre- and post-compensation–Because bit shift can be predicted, it is possible to compensate–correct–for this shift within a controller. The two methods that are frequently employed are precompensation, and postcompensation.

With precompensation, bits are deliberately shifted in the direction opposite that of the expected shift during the write phase of the cycle. Hence so-called Write precomp.

Postcompensation, on the other hand, alters the read signal. Each bit is considered a function of the read channel frequency response and the signal is changed to compensate for any bit shifts.

Rail–is the milled part of a slider that supports the head assembly on the flexure, and provides an air bearing surface to permit accurately controlled flying. Moreover, the name applies since the effect of the surface is very similar to that of a railroad track in that it provides support and gauge.

Seek time–A head mechanism must travel a finite distance from one track to another in a 48 tpi drive, the longest distance is from track 0 to track 35 a full stroke of about 0.8-in., for example. Typical specifications give the 'seek time' in terms of track to track, and track 0 to the last track.

Settling time–Once the head moves into the proper track, it must settle both mechanically and electrically. Since drive manufacturers are aware of the unique characteristics of the drive they make, they take this in account in the onboard electronics. However, controller designers must pay attention to it in order not to expect a data stream before the head is ready.

Slider–Is the diced substrate form the read/write head geometries are bound to. The slider is made up of a block of silicon, or in the case of thin film heads, ceramic material which has been milled to form rails, and either has cores deposited or glass bonded to the ends. Interestingly, this block 'slides' into the flexure mount and is designed to 'slide' over the medium.

Track Density–Specifications define the number of tracks per inch that the drive exhibits. These numbers can range from 48- to 600-tpi depending on the class of the drive. Simply stated 'track density' is the number of tracks per unit length. Notice though, that in a typical spec the tpi is less than the bpi by several factors. Remember one is referencing a physical size governed by head geometry while the other is electrical and although partly a function of the head gap size is magnitudes greater in density. (See Areal density).

Winchester–May conceivably be the most misused word in current usage. It refers to 3340 technology developed by IBM (circa 1973) that employs a sealed disk and positioner assembly. The name came about because it was first dubbed 3030 technology, or so insiders say, and as a consequence the recording method picked up the moniker of the famous rifle maker.

Gigabytes on the horizon

Although a great deal has happened in the last several months in storage technology, expect to see even more as we get deeper into the eighties.

Reports have it that Control Data already has the capability of fielding a 200 Gbyte single platter 8-in. optical disk, and that Optimia, a subsidiary of Shugart Associates, has similar capability. Therefore don't be overly surprised to see early introductions of high-performance optical systems by late this year. In addition, you might keep your eyes open for a newly formed company in Irvine CA that is supposedly working on an optical system in a 5.25-in. form factor. Even if the product surfaces this year, it will more than likely be far too expensive to plug into your TRS-80.

Financial Accounting On The Computer

Steve Patchen and John Snow

BOSS Financial Accounting System
Lifeboat Associates
1651 Third Ave.
New York, N.Y. 10028

BUSINESSMASTER Accounting Software
O.E.M. Software
18051 Crenshaw Blvd. Suite F
Torrance, Ca. 90504

In this article we will discuss some of the important ways in which packaged financial accounting software differs and in which it might fail or succeed in meeting the financial accounting needs of the business. As an aid to this discussion, we will compare two existing systems which run under CP/M-80. On the whole, neither of these accounting systems is necessarily any better or worse than other systems but they contrast well with each other on points important for a system to meet a particular application. Accounting systems are usually very similar because there is some standardization in accounting practices. This is due in some part to the pressure of federal and local tax laws. The influence of tax requirements is most strongly felt by small businesses which might otherwise be run from some simple check ledger accounting system. As a business becomes larger, financial planning and budgeting of money handled by several people become important. The type of business and the manner in which business is conducted also have a strong influence upon the structure of the accounting system. Thus whether the company is a corporation or a sole proprietorship; whether it manages property, manufactures and sells products or provides services makes a difference in the requirements of the accounting software.

The structure of the chart of accounts of the general ledger shows many differences. Likewise, the source of financial data posted to the ledger can be other automated subsystems or a manual transaction entry system. The volume of transactions conducted by the business in such areas as payroll, accounts receivable, cash disbursements and inventory can make a big difference in the suitability of a particular software package. Indeed, this volume may make a subsystem like payroll, which requires fast turnaround from inputs to check disbursement, more important than the general ledger system itself. The ability to add additional modules to the system at a later date may also be important to a business which already has part of its accounting automated. This would require that the data generated by the first module be accessible to the module added later; the first module would have to be able to utilize data generated by the later module, if it replaces a manual system previously used to provide information to the first module.

As a result we have changed Table III to list outstanding accounting functional modules and have retained the other tables used in the database evaluation articles.

Distribution and Support

Because financial accounting is closer to particular business requirements than the general functions provided by database systems, the method of distribution and support become more important and more closely related to those individual business needs. The BOSS is distributed exclusively by Lifeboat Associates and is available only in object code to run under CP/M-80. The BUSINESSMASTER is distributed in CBASIC source code to OEMs and distributors, to be supported locally by them. (Mail order houses are also distributing it unmodified at dirt cheap prices but this somewhat defeats the value of it as an OEM product, because it is difficult for a novice to implement and use it.) It is difficult to get good support for particular business needs from mail order houses because they tend to spend update and development effort mostly on those things which affect a lot of customers. The cost of the support is cheaper; that is updates usually cost $25-$30, but there is usually little time to work on individual requirements not met well by the current state of the software. Local support for a product can usually be tailored more to the individual business needs, but it costs more because fewer customers bear the burden of the cost of the modifications to the software.

The $2500 price of the BOSS is comparable to Peachtree, Structured Systems, Graham-Dorian, and others in that exalted price range. At the other end of the price scale are TCS, OEM, and the Osborne packages. These cheaper packages sell for $100 to $400. The quality of the documentation and other cumbersome features are reflected in the lower prices, and thus these systems are not recommended for the inexperienced computer user.

Functions and Features

The BOSS is a fully developed accounting and financial reporting system. This software requires little computer knowledge and only rudimentary bookkeeping skills. You do have to know the difference between debits and credits, but the BOSS even has a refresher section to clarify their use. The BUSINESSMASTER uses only +'s and −'s during entry but shows them as debits and credits on screen forms and reports. The BOSS has accounts receivable and accounts payable modules with several reports associated with each, including the ability to print checks. The receivables cannot produce an invoice and the payables cannot produce a purchase order, however. BUSINESSMASTER is capable of producing invoices, purchase orders, and checks. It does not print statements. Both systems provide depreciation schedule facilities. The BOSS also can produce loan amortization schedules.

There is one major functional flaw in the BOSS: It doesn't contain a payroll module. Balcones is working on one but the current system is incomplete without it. Any small businessman has to do three essential things: deliver his product, hustle his money and make payroll on time. The BOSS can help him keep track of his finances but it cannot help get the payroll out. If you only have a couple employees or if you are using a service bureau for payroll currently then you can probably live with this shortcoming. The BUSINESSMASTER has a payroll module set up for federal and California state taxes. It must be modified for other states.

The BOSS general ledger will handle 900 accounts. The account numbers are 3 digit integers. Three digits are sufficient for most applications. (A notable exception occurs when the business uses a large inventory system requiring ledger entries for many inventory categories. It is usually desirable to use the same numbering in both the ledger and the inventory.) The accounts payables and receivables can handle 9000 vendors/customers. You can have 900 departments, each cost centered and 900 categories of products or salesmen. You can also set up any number of companies. If you have an accounting firm this program will handle all of your accounts. It will give you a bewildering array of financial reports, such as liquidity, return on equity, return on total assets, etc., a very impressive shopping list. All report selection is menu driven.

The BUSINESSMASTER general ledger will handle 9999999900 accounts. The account numbers are 10 digit integers. The customers and vendors also have 10 digit account numbers. There is no departmentalization except by account number variation. Different companies can be set up by making up separate data diskettes for each. The number of reports available is limited compared to the BOSS.

The BOSS can use NEBS forms for checks and statements. This is thoughtful. It can save time in designing forms. The reports are available anytime; closing procedures are not necessary before printing a report. Summary reports can be run for individual departments within a company. This makes performance reviews convenient. Customer files can be brought up on screen at any time for reference or corrections. Thus you have access for phone conferences with customers. You can refer to records by ID codes or alphabetically and can scan forward and backward from

TABLE I
Facts & Figures

Package or Version name:
(Name of the versions reviewed)

The BOSS Financial Accounting System Version 1.08

The Businessmaster Accounting Software (no version given)

Price:
(List price or Lifeboat price)

The BOSS lists at $2495.00 from Lifeboat, the exclusive distributor.

The Businessmaster is sold by O.E.M. Software for $375 complete with source code and distribution rights. Mail order prices for end users vary from $100 to $250.

Systems available for:
(Computer or operating systems on which the package can operate.)

Both packages run under CP/M-80.

The BOSS provides its own runtime root and is written in Microsoft compiler BASIC.

The Businessmaster requires CBASIC.

Memory requirements:
(RAM memory required.)

The BOSS requires a 54k system or more.

The BUSINESSMASTER will run on a 48k system.

Printer Requirements:

BOSS requires a 132 column printer but can use compressed printing on a narrower printer.

BUSINESSMASTER uses less than 80 columns for most reports. But 132 columns is required for the payroll ledger and the inventory reports.

Diskette capacity required:
(Minimum amount and amount required for for each of the test applications.)

BOSS requires > = 200k drives, two or more

BUSINESSMASTER will run on 160k drives after it is compiled.

(continued next page)

TABLE I (continued)

Utility programs provided:
(Programs provided other than those required for routine use of the package.)

BOSS provides a backup utility.

Portability:
(How difficult is it to move the application and/or the data to another package or computer.)

BOSS can be moved to other Z80 or 8080 CP/M-80 systems.

BUSINESSMASTER can be moved to any machine which can run CBASIC. i.e. 8086 cpu's also.

User skill level required:
(i.e., novice, amateur or professional skill level required to use and/or implement each application using the package.)

The BOSS can be setup and used by a novice.

The BUSINESSMASTER should be setup by a professional and training should be provided for users.

System upgrade policy:
(Is an upgrade subscription available? Cost? Who provides? If no subscription available, usual cost of upgrades.)

BOSS updates are available only from Lifeboat Associates.

BUSINESSMASTER requires local support.

the current record with simple keyboard control characters. The amortization routine requires all four factors of principal, interest rate, duration of the loan, and the amount of the payment. Most other systems only require three of the four and compute the missing one.

The BUSINESSMASTER uses NEBS forms only for the checks. It has a mailing system which utilizes the customer file. This is convenient for many businesses. There is a special utility to clear the payroll file amounts at the beginning of a new year; it must be run outside the menu system. There is a similar routine for the general ledger, allowing the reposting of the journals as a check or after data damage of some sorts. The essential reports for each module are provided, but there are not any of the useful financial management ratios and earnings reports such as those provided by the BOSS. Only one minor bug was uncovered during use of this system. Some reports failed to turn the printer off if the report happened to end exactly at the last printable line on the page.

A three level security system is included in the BOSS to restrict sensitive parts of the system to authorized personnel. BOSS also does an integrity check every time it loads a program module. This eliminates the possibility of a bad program module scrabbling data. No security is provided for the BUSINESSMASTER system. Thus data integrity is more threatened by operator error and equipment failure.

Setup and Use

Both systems run under CP/M-80. The BOSS requires 48k in the transient program area, i.e. usually a 54k system. The BUSINESSMASTER will run under a 48k system. You will need a 132-column printer for the BOSS or one with compressed print like the Epson MX80. During installation you select the compressed print option so that the reports will fit in 8 1/2 inches of width. A minor modification is required if you have a Centronics printer. The BUSINESSMASTER prints most of its reports on 8 1/2 inch paper. The exceptions are the payroll register and the inventory reports. You need at least 200k on the system device for the BOSS. BUSINESSMASTER requires less to run but must then be compiled on a different machine.

BOSS performs as advertised. It won't scare your secretary or you either. After installation, which is explained step-by-step, the system is discussed menu-by-menu from the orientation through day-to-day use. You do have to spend some time acquainting yourself with the organization of the material in the manual and with the organization of the system, but this is not difficult. You should run through the tutorial and read the special sections on how to speed up data entry and how to print reports. The language in the manual is business oriented. If you are installing the system yourself you will need to be familiar with CP/M-80 and PIP in order to transfer files to the disk configuration you desire. The step-by-step instructions are clear to the user who has no knowledge of computer jargon.

Setup and operation of the BUSINESSMASTER are discussed in the manual. You are given guides as to which data files have to be on which diskettes and an example is provided to demonstrate setting up a new chart of accounts. A field called a TAG is used by the chart of accounts and some other modules. It is discussed a little in a few places in the manual. But the explanation is far from clear. It took a considerable amount of time for me to determine the exact requirements for the use of the TAG in the various files.

The terminal installation for the BOSS is easy for most common terminals.

The installation procedure for the BUSINESSMASTER requires changes to a CBASIC source file which is included in most files during compilation. This could not be done by someone not familiar with CBASIC. This also means that a compiled version of the system can only be run on other systems with the same terminal.

Documentation

The BOSS manual is excellent. It is the ultimate in detailed explanation. There is a comprehensive and easy to follow installation section, a good tutorial section, and illustrations for each type of entry. There are special help sections; one for problems which might be encountered, one for speeding up data entry and one for reviewing basic bookkeeping fundamentals. Unfortunately there is no index, even though there is a tabbed divider for one. The reading might seem dull, but once you get through it far enough to grasp the operation of the BOSS, it really shines. There is an obvious mistake in one of the examples for the debit/credit tutorial section. A $1000 charitable donation is proposed for a sole proprietor; this is not only impractical, it is illegal. An individual can make such a contribution but it cannot be taken as a business expense.

The BUSINESSMASTER manual is bad. The data file contents and the menus are discussed and some of the setup requirements are covered, but the manual is full of holes. These explanations are far from clear, especially to someone not familiar with computer structures and jargon. It would be impossible for a novice to extract the information he would need to get the system running. This system definitely requires technical support and local training. There seems to be some leeway in laying out the data files because some requirements are stated, though the reasons for them are not. It is not at all clear from the manual, however, what variations are possible.

TABLE II:
Qualitative Factors

	Rating*	
	BOSS	MASTER
Documentation		
organization for learning	7	3
organization for reference	5	3
readability	7	2
includes all needed information	6	1
Ease of use		
initial start up	7	4
conversion of external data	1	4
application implementation	4	6
operator use	7	4
Error recovery		
from input error	7	4
restart from interruption	7	4
from data media damage	7	4
Support		
for initial start up	6	*
for system improvement	4	*

*Ratings in this table will be in a 1-7 scale except where local support is required. where:

1 = clearly unacceptable for normal use
4 = good enough to serve for most situations
7 = excellent, powerful, or very easy depending on the category

Backup, Error Recovery and Data Organization

Another good feature of the BOSS is that it opens files only during the momentary write time. If you lose power or the hardware fails, you only lose that single transaction. All the other records are secure on the magnetic media. The BOSS also has an integral backup system. That is, it doesn't require the use of PIP. The operator runs it by menu control and by specifying the drive to place the backup files on. The BOSS is also able to quickly look up the titles for account numbers and the customers and vendors by their numbers during transaction entry without noticeably slowing down entry because it uses B-Tree indices. (See the discussion of indices in the Dec 81 issue, p15. and the corrections to that article in this issue.)

Review Summary

The BOSS is an excellent choice for the novice computer user if he does not require complete financial accounting. The BUSINESSMASTER is a good choice if a more complete financial system is required and local support is available. The local support should include experience with CBASIC and familiarity with state and city tax requirements.

John Snow is an accountant and business consultant. He has run his own firm for sixteen years. In 1980 he established Small Business Computers in the Detroit area.

Steve Patchen is a data engineer who provides hardware and software support for small business computer users in Southeastern Michigan. He has run LAB Data Systems in Pontiac, Mich. for 4 years.

Please send all correspondence to Steve Patchen in care of *Lifelines*, 1651 Third Ave., New York, N.Y. 10028.

(see next page for TABLE III)

TABLE III
Data Management Capabilities
& Accounting Functions

	BOSS	MASTER
A. Underlying Data Structures		
1. indexing	YES	NO
2. files accessible by other programs without professional help	NO	YES all ASCII
B. Functions Provided		
1. general ledger	YES	YES
a. ledger trial balance	YES	YES
b. ledger summaries	YES	ONE
c. income statement	YES	ONE
d. balance sheet	YES	YES
e. comparative reports	YES	ONE
2. payroll	NO	YES
3. accounts receivable	YES	YES
a. invoices	NO	YES
b. statements	YES	NO
4. accounts payable	YES	YES
a. purchase orders	NO	YES
b. check printing	YES	YES
5. inventory	NO	YES
6. mailing system & labels	NO	YES
7. financial calculations		
a. amortization	YES	NO
b. depreciation	YES	YES

The Undocumented "CALL" Instruction in dBASE II Version 2.02

In the December issue of Lifelines Carl Warren mentioned a "call" instruction in dBASE which would allow interfacing dBASE "CMD" files with assembly language routines. George Tate of Ashton-Tate was kind enough to supply details of how to use this feature. An example follows:

```
store "I am a subroutine" to mvar
call mvar
```

On executing these lines, dBASE vectors to location A400h with the HL register containing the address of the first byte of the memory variable. This byte contains the length of the variable, and the bytes following it contain the variable itself. You may use all registers and alter any memory location above a400h, returning with a "ret" instruction. The only time dBASE will use this memory area is during a SORT, when it will employ a400h up to BDOS to sort in memory. The subroutines to be called must be loaded into memory before dBASE is brought in.

The new version of dBASE, now in beta testing, will have several extensions to allow subroutine calls, e.g. "SET CALL TO CALLADDRESS", PEEK, and POKE.

Renew

Did your subscription begin in March 1981? If it did, you've probably received two reminders from us and you should know that renewal time is here. If you don't renew now, you'll almost surely miss our big March issue, which will include some exciting reviews and product information.

Don't miss out on your sole source of information on what's really happening with CP/M-80 and related software. Call or write the *Lifelines* Subscription Dept., 1651 Third Ave., New York, N.Y. 10028. Telephone: (212) 722-1700.

8080 Assembler Programming Tutorial: Control of the Execution Sequence

Ward Christensen

All the instructions previously used were executed sequentially. In this section of the tutorial, I'll cover the instructions which allow changing the execution sequence. Some, such as JMP, CALL, and RET, are unconditional. Others are conditional.

To make the examples more meaningful, I will cover the conditional examples after going into the arithmetic and logical instructions.

Let's start with the simplest instruction for altering the execution sequence:

JMP

The JMP instructions tells execution to continue at a specific place in the program, rather than continuing sequentially. The format of the JMP command is:

JMP address

Typically, "address" is a LABEL in the program.

The following example shows the use of the JMP instruction: to repeat some process "forever".

```
;
;program example for use of JMP
;
    LXI  H,XYZ ;point hl to XYZ
LOOP MOV  A,M  ;get character
    .
    .          ;do something
    .
    INX  H     ;to next char
    JMP  LOOP  ;loop forever
```

Are there alternatives to the JMP instruction to change the execution sequence of a program unconditionally? Only one way:

PCHL

"Big" computers typically have a way to "jump to the address in a register". The 8080 shares this function in only one small way: the ability to "jump" to the address in the HL register.

It accomplishes this function by "moving" HL to the program counter (PC). Thus, the operation code: PCHL. It stands for "Program Counter, from H and L". It has no operands, nor conditional variations.

CONDITIONAL JMPs

JMP transfers control unconditionally. To JMP only if some condition is met, you use the conditional JMP instructions. The conditions tested are the bits of the PSW as we discussed earlier. The conditions which may be tested, and their abbreviations are as follows:

Z	zero
NZ	not zero
C	carry set
NC	carry not set
M	minus (negative) result
P	zero or positive result
PE	even parity result
PO	odd parity result

The instructions for these conditional JMPs are formed by the letter J, followed by the 1 or 2 letter abbreviation shown. Note there might be some surprises here: for example, it is easy to remember that "P" means positive, but not as easy to remember that "positive" in the 8080, **included zero**! Thus, a JP (JUMP Positive) instruction will jump on positive, or zero. With these conditional JMP instructions, you can perform "looping" which is like the familiar FOR - NEXT loop of BASIC. For example, if you want to do something 10 times:

1-put the value 10 in some register;
2-perform the operation;
3-decrement the register;
4-"test" to see if the register is 0, with a JNZ instruction.

This is possible because in step 3, you code a DCR which sets the zero bit in the PSW if the register is decremented to 0. Then, the JNZ (jump if not zero) can test the zero condition:

```
         .
         .
    MVI  B,10  ;INITIALIZE COUNT
LOOP
         .
         .     ;SOME INSTRUCTIONS
         .
    DCR  B     ;DECR. LOOP COUNT
    JNZ  LOOP  ;LOOP IF MORE
         .
         .
```

Note how the B register was used for the loop count.

An aside on good versus bad program commenting: note my use of comments. The comments say "why" I coded the instruction. If on the "DCR B" instruction, I had put a comment ";DECREMENT B", I would not be telling you *why* I was doing the instruction. "An assembler program's comments should explain the *logic* of the program, not explain the instruction set". An obvious exception would be programs or program fragments specifically intended to *teach instructions.*

With the JMP and conditional JMP instructions, you can see how the flow of control in a program may be changed.

Another major facility available relates to going to a program, and being able to return from it no matter where it was called from. Read on...

CALL and RET

Let's say I wrote a routine which multiplies two numbers. If I want to execute that "subroutine" from several places in our program, then using JMP will not work. It would be nice to have a means to go to the routine, and return to where we came from. BASIC uses GOSUB to go to a subroutine, and RETURN to come back.

The CALL instruction is like a JMP, except that the address of the instruction following the JMP is placed on the

(continued next page)

stack. Thus if I am at address 100, with the stack pointing to 300, and I issue a call to address 200, the following will happen:

1-The address of the instruction following the CALL (i.e. 103) will be "pushed" onto the stack.
2-The program counter will be changed to 200, therefore execution continues at address 200. A JMP 200H would also go to 200, but it wouldn't result in the next instruction's address being placed on the stack.

When the subroutine at 200 is finished, it should go back to 103H. The RET instruction has that function. RET "pops" the stack into the program counter. Thus execution resumes at 103H.

The program now looks like this: I added addresses to the left margin, although they wouldn't normally appear in any program you write until the program was run through the "assembler" which converts your instructions into machine language.

```
    ;ASSUME THE STACK IS 300H
           .
           .
100        CALL MULT
103        .          ;RETURN HERE
           .
           .
    ;
    ;MULTIPLY SUBROUTINE
    ;
200 MULT   .
           .
           .
2XX        RET        ;RETURN
                      ; TO CALLER
```

In typical use, the CALL and RET instructions may be "nested", i.e. one routine can call a second, and it, in turn, can call a third. We'll see an example shortly, but first:

RST

Think of RST (Restart) as a very special CALL instruction. Like CALL, the address of the instruction following the RST is placed on the stack. However, RST may only go to one of 8 fixed addresses in memory:

This instr	calls:
RST 0	0 (00H)
RST 1	8 (08H)
RST 2	16 (10H)
RST 3	24 (18H)
RST 4	32 (20H)
RST 5	40 (28H)
RST 6	48 (30H)
RST 7	56 (38H)

RST is generally thought of as an "interrupt related" instruction. More details on this in a later tutorial. In programming an 8080, you are most likely to encounter RST when testing programs under DDT (Dynamic Debugging Tool) or SID (Symbolic Instruction Debugging). Both of these use the RST 7 instruction to do functions like tracing and breakpointing. To do so, they place a JMP at location 38H. You can take advantage of this by coding RST 7 when you want to return to DDT or SID after testing your program. There will be more details on program debugging in future parts of the tutorial.

Let's return for a minute to the PCHL instruction. Since it is effectively a JMP to the address in HL, there might be times you want to, instead, CALL the address in HL (but admittedly, not very often).

Here is how you would do that: you have to use the CALL instruction — there is no getting away from it. However, what *it* in turn calls, is a PCHL instruction. Thus, the return address is set up by the CALL, but control transfers in a hop-scotch form, first to the PCHL instruction, then to the address contained in HL. The program to CALL the address in HL would now look like:

```
        .
        .
        CALL GHL
        .
        .
        RET
GHL     PCHL
```

The CALL GHL puts the address of the instruction following CALL on the stack, and transfers to the label GHL. GHL then executes the PCHL, and branches to the address in HL.

I chose the name "GHL", and not "PCHL" because the CP/M assembler ASM does not allow an operation code (PCHL for example) as a label. In later sections of the tutorial, I'll show use of PCHL in branching to addresses within CP/M, so as to work on virtually any system.

CONDITIONAL CALL and RETURN

Just as there were conditional variations of the JMP instruction, there are conditional variations of the CALL and RET instruction. Conditional CALL instructions are formed by the letter C followed by the condition abbreviation, and conditional RETurn instructions by the letter R followed by the the condition abbreviation.

SUMMARY

Here is a table in which I have spelled out all of the instructions relating to control of the execution sequence:

		(unconditional)	
CHL and	JMP	CALL	RET
		(Conditional)	
zero	JZ	CZ	RZ
not zero	JNZ	CNZ	RNZ
carry	JC	CC	RC
no carry	JNC	CNC	RNC
minus	JM	CM	RM
positive/0	JP	CP	RP
parity even	JPE	CPE	RPE
parity odd	JPO	CPO	RPO

SAMPLE PROGRAM

Let's put this all together into a partial program: the program will print a message. It has two subroutines, PRTM to print the message, and PRTC which prints a single character. (The short labels are a result of squeezing the programs into these narrow columns).

In order to avoid instructions which haven't been covered, I will assume that the message has a length byte in front of it. It tells how many characters to print. Later on, I'll show the more common technique of storing a special character at the end of the message, so you don't have to know how long it is in advance. Here is the program:

```
     LXI  H,MSG ;point to msg
     CALL PRTM  ;print the msg
       .
       .
;
;define the msg,
;
MSG  DB   18     ;msg length
     DB   'This is a message'
     DB   0DH,0AH ;CR/LF
;
;MSG print subroutine
;
PRTM MOV  B,M    ;GET LENGTH
PRTL MOV  A,M    ;GET MSG CHAR
     CALL PRTC   ;PRINT THE CHAR
     INX  H      ;TO NEXT CHAR
     DCR  B      ;COUNT DOWN
     JNZ  PRTL   ;LOOP TILL DONE.
     RET         ;RETURN
       .
       .
PRTC   .
; (We'll discuss the output in-
;  structions needed by this
;  routine, at a later time)
;
; Note that this program must
; preserve the BC register which
; has the count in it.
    RET          ;RETURN
                 ;  FROM PRTC
```

CONTROL OF THE EXECUTION SEQUENCE

SECTION 2
LOGICAL AND COMPARE INSTRUCTIONS

JMP, CALL, and RET control the execution sequence. Their conditional variations give real "power to programming".

In this section, I'll cover the "logical and compare" instructions, most of which relate directly to conditional execution: they are the instructions that *set* the conditions.

THE LOGICAL INSTRUCTIONS

The logical operations available on an 8080 support ANDing, ORing, EXCLUSIVE ORing, COMPLEMENTing, and bit SHIFTING. Recall from the TERMS section of the tutorial, that when you AND two bits together, the resulting bit is one only if *both* of the original bits were one.

When you OR two bits together, the result is a one bit if EITHER of the original bits were one.

When you EXCLUSIVE-OR two bits together, the result is a one bit if ONLY 1 of the original bits were one, the other zero. This is similar to OR, but *excludes* the case of where *both* bits were one.

This might be easier to follow using the following table. Since there are only 4 possible combinations of 2 bits (both on, both off, first on second off, and first off, second on), the table will use 4 bit numbers, allowing for all possible combinations:

AND	OR	EXCLUSIVE-OR
1100	1100	1100
1010	1010	1010
1000	1110	0110

The most commonly used operation is AND. Why? Because the AND operation has the property of "turning off" unwanted bits in a byte, and therefore "isolating" only the wanted bit or bits.

For example, when you are testing to see if a key has been pressed on your keyboard, you can use the IN instruction to input an 8-bit byte. In this byte, usually called the STATUS byte, a single bit will go either on, or off, depending upon the type of keyboard interface, indicating the character is ready. Suppose your keyboard has the bit in position 40H (Hexadecimal) go on when a key has been pressed. In binary notation:

0x000000

the "x" bit goes on.

The other bits (shown as 0 above) may also be used to indicate something. For example, another bit could show that an output device, say a display terminal, is ready to receive another character. Thus we want to isolate just the 40H bit. If we input this status, AND it with 40H, we will zero all the bits except the 40H one. The zero bit in the PSW will then reflect the state of the 40H bit. Thus, we can do a conditional jump (JZ, jump zero, or JNZ, jump not zero) to test if the bit is off or on.

Here's an example. "B" is the bit we want to test, and "X" are the bits we don't care about:

Input Port:	X	B	X	X	X	X	X	X
"AND" with:	0	1	0	0	0	0	0	0
Results in:	0	B	0	0	0	0	0	0

Thus "ANDing" "isolated" just the bit we care about.

Let's look at the logical instructions themselves. First, the immediate instructions.

ANI, ORI, and XRI

These instructions (AND immediate, OR immediate, and EXCLUSIVE OR immediate), combine the 8 bit value in the accumulator (A-register) with an 8 bit value from the instruction itself. The result is placed back in the accumulator.

Typically the value in the instruction is coded as a hexadecimal value, for example:

```
ANI  40H
```

In general, the format of the instruction is:

```
ANI  value
ORI  value
XRI  value
```

You will often want to set up a specific value advance, via an EQU (equate) instruction. Thus:

```
MASK EQU  40H  ;INPUT STATUS
       .
       .
     ANI  MASK
```

The ANI MASK is actually an ANI 40H, but it is more readily changed if the value of MASK is used in several places, or if you don't want people to have to "dig" for it in your program.

ANA, ORA, and XRA

These 3 instructions, AND with accumulator, OR with accumulator, and EXCLUSIVE OR with accumulator, perform the same as the immediate logical instructions. However, the second part of the data comes from

(continued next page)

one of the registers or from memory, instead of from "immediate data" in the instruction itself. The instruction format is:

```
        ANA   reg
        ORA   reg
        XRA   reg
```

where "reg" is one of A, B, C, D, E, H, L or M for memory addressed by HL.

Thus, if you want to AND the accumulator with a 40H, you could do it in either of 2 ways: ANI 40H, or you could have put 40H in some other register, say B, via MVI B,40H — then you could:

```
        ANA   B
```

This technique looks more complicated, but it executes just a little faster, and if you are going to be in a loop testing for a bit to come on from some device attached to your computer, you might want to be able to do something very quickly when a bit comes on.

If microseconds (millionths of a second) count, the ANA with a value in a register might be better. Another reason for using ANA over ANI occurs if the value is not known at the time the program is assembled. It would be loaded into the register at execution time after being calculated. You could have stored a value into the second byte of an immediate instruction to change it's value. Why? because the listing might say "ANI 40H" but by storing into the instruction, it might be "ANI 20H" at execution time. Makes debugging harder, and leaves room for error when modifying the program later. These instructions also have some subtle and interesting side effects. Specifically:

```
        ORA   A
          or
        ANA   A
```

does not change the contents of the A register (because ORing or ANDing something with itself leaves it unchanged). However there are two results: First, the Program Status Word (PSW) bits are set to the value, and second, the carry bit is set off.

Why is it useful that the PSW bits are set? Well, suppose you are scanning through a series of bytes (say an ASCII message) and wanting to test for the end of the string of characters. An easy way would be to put a byte of 0 at the end of the message. It is logical to code:

```
MOV   A,M        ;get a char.
CPI   0          ;Is it 0?
JNZ   LOOP       ;..no, loop
RET              ;..yes, ret
```

However, you could replace the "CPI 0" which is a 2 byte instruction, with "ORA A" which is a 1 byte instruction. JNZ LOOP still stays, since the zero indicator will be set properly in either case. (Details on ORI are coming soon).

As to ANA A and ORA A setting off the carry bit: the carry bit is sometimes used as a "flag" indicating success or failure of a routine. For example, after a CALL, you could do a JC ERROR meaning the routine called had an error, and set carry.

A simple instruction, STC, is used to set carry. There is no corresponding instruction specifically designed to turn off carry. However, using:

```
        ORA   A
          or
        ANA   A
```

turns off carry. My preference is ORA A, but both have *exactly* the same effect.

XRA A is also interesting, in that it zeros the accumulator. This is faster, and takes one less byte of instruction than "MVI A,0". Another difference, which you might want to make use of some time, is that XRA A turns off carry, while MVI A,0 doesn't change it.

CMA

CMA (complement accumulator) "flips" each bit in A. For example, the contents of the A register:

Before CMA: 10110101
After CMA: 01001010

Note that this DOES NOT make A the "negative" of what it was. For example:

Before CMA: 00000010 = 2
After CMA: 11111101 = −3

You can see that 2, via CMA, becomes −3, not −2. If you do want to make the value in a negative, you must add 1 after CMA. The easiest way is:

```
CMA              ;flip bits in A
INR   A          ;then make it
                  negative
```

The result of these two instructions is called the "two's complement" of the original number. There is also a name for the result after just the CMA: "ones-complement". To review: the ones-complement of a number is obtained by simply flipping the bits. The twos-complement is formed by flipping the bits, then adding one.

The following instructions could be considered to be part of the "logical instructions" in that they deal with the BITS of A.

RAL RAR RLC RRC

These 4 instructions ROTATE the bits in A. Those with L rotate left, those with R rotate right. For example, if the accumulator contains:

00001111

and we execute a RRC instruction, it will now contain:

10000111

The rightmost bit will also be copied into the carry bit, i.e. it will now be a 1. More generally, RRC and RLC rotate the 8 bits of the accumulator, and copy the bit which "rotated around" into the carry bit.

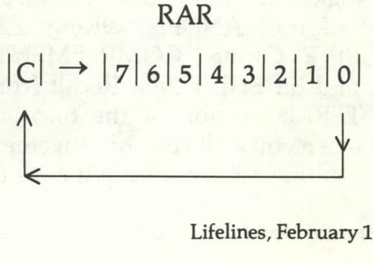
RAR and RAL also rotate the accumulator, but they treat it as a 9 bit value. The 9th bit is the carry bit. Thus if carry is on, and A contains:

00001111

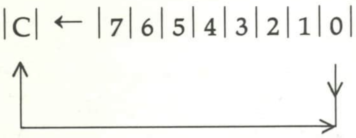
then a RAR instruction makes A:

10000111

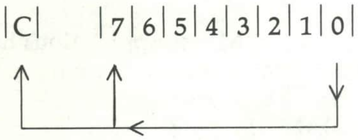
and will result in the rightmost 1 bit being shifted into carry, leaving it on. Pictorially:



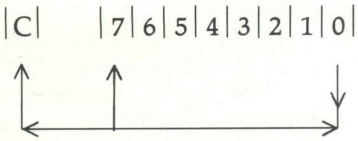
RAL



RRC



RLC



COMPARE INSTRUCTIONS

Single comparison can always be done using subtraction. For example:

SUI 'R'

will SUbtract Immediate, an ASCII 'R' from the accumulator. If the accumulator contained an 'R', the zero indicator will be set, so you can "JZ GOTR" i.e. jump to "GOTR" if the value was an R. However, we have CHANGED the value in A, so we cannot then:

SUI 'G'

to see if it might have been a G. It would be nice to have a subtract instruction that *doesn't* change the value. Read on...

CPI

CPI does a "compare immediate" of the accumulator, with an immediate 8-bit value. For example:

CPI 'R'

compares the accumulator with an ASCII R, setting the zero indicator (allowing "JZ label") if the value was R. Unlike the SUI instruction, the value is A IS NOT CHANGED by the CPI instruction. This permits:

```
    CPI   'R'      ;is it R?
    JZ    GOTR     ;yes
    CPI   'G'      ;..or G?
    JZ    GOTG     ;yes
etc.
```

CMP

The CMP (compare) instruction also compares 8 bits in A, but to a register rather than immediate data. The format of the instruction is:

CMP reg

where "reg" is one of A, B, C, D, E, H, L or M for memory as pointed to by HL.

ANDing and ORing "like a high level language"

I have covered the control of the execution sequence, and the logical and comparison instructions that are used to set the conditions. I would now like to relate these to, say BASIC usage of AND and OR. Consider the BASIC program fragment:
30 IF X=2 OR X=3 THEN GOTO 50
Do you know what goes on "under the covers" in BASIC, when it executes these instructions? The popular Microsoft BASIC first computes the result of "X=5", then the result of "X=2". It then "OR"s these two results, and makes the "THEN GOTO" decision based upon the results. You can try it by trying "pieces" of the line, and observing the results:

```
A>OBASIC
33273 Bytes free
BASIC Rev. 4.51
[CP/M Version]
Copyright 1977 (C) by Microsoft
Ok
LET X=2
Ok
PRINT X=3
 0
Ok
LET X=3
Ok
PRINT X=3
 -1
Ok
```

"Under the covers", MBASIC is working with "bits". Printing "X=3", when X was not equal to 3, showed 0. The result when X was equal to 3, was "−1". The value of −1 is actually "FFFF", or "all bits on". You can see for yourself:

```
PRINT HEX$(X=3)
FFFF
Ok
```

Let's get back to assembler! (...and try something similar). Suppose you want to jump to some label if the accumulator contains either 2 or 3. You would not likely perform the same logic as BASIC did. As a matter of fact, you don't even use any "or" instructions. Instead, you do the "or" in your head, as part of the logic of designing the program. What? Well, let an example show:

```
CPI   2      ;IS A=2?
JZ    XXX    ; YES, JMP TO XXX
CPI   3      ;IS A=3?
JZ    XXX    ; YES, JMP TO XXX
```

There are alternatives, such as by putting the routine XXX next in line in the program:

```
        CPI   2     ;IS A=2?
        JZ    XXX   ; YES, JMP TO XXX
        CPI   3     ;IS A=3?
        JNZ   NOTX  ;NO, SKIP XXX
XXX:    .
        .
NOTX:
```

As you can see, although we wanted to "jump to xxx if A was 2 or 3", no "OR"-type instruction was used.

Let's code that yet another way. This technique will be useful for comparing a "range" of values. For example, it can test if A is any value from 2 to 8, in the same number of instructions as to test it for the values 2 or 3.

I will use CPI. It is easy to use CPI to test if a value is equal to, or not equal to, some value. I showed this in the previous program example, using JZ and JNZ. Its time to resort to "gimmicks" - a way to remember something that can't be "figured out" each time you need it.

The "gimmick" is: "C.A.L". It refers to the state of the PSW carry bit, after doing a compare instruction. C.A.L. stands for "Carry if Accumulator is Lower".

An example:

```
MVI   A,2
CPI   2
```

The PSW "zero" indicator will be set. However, "carry" will not be. In the case of:

```
MVI   A,2
CPI   3
```

(continued next page)

"carry" will be set. "C.A.L.", or "Carry if Accumulator is Lower" — and the accumulator, (2) is lower than what is being compared, the immediate 3. Back to comparing a range of values: again, a program to test if the accumulator contains a 2 or 3:

```
        CPI   2       ;IS A < 2?
        JC    NOTX    ; YES, SKIP XXX
        CPI   4       ;IS A < 4?
        JNC  NOTX
XXX:    .
        .
NOTX:   .
```

To compare if A is any value between 2 and 8, just change the "CPI 4" to a "CPI 9". As long as I am covering alternatives, there are even more. The DCR instruction can be used to test for small values, since decrementing a register to zero sets the PSW "zero" flag:

```
        DCR   A       ;SET ZERO IF..
        DCR   A       ;..A WAS 2
        JZ    XXX     ;JMP IF IT WAS 2
        DCR   A       ;SET 0 IF A WAS 3
        JNZ   NOTX    ;SKIP IF NOT 3
XXX:    .
        .
NOTX:   .
```

A few last thoughts on the use of compare instructions, carry, and "C.A.L". First, let me relate it back to BASIC programming again. If you use CPI or CMP, then JC, consider the parallel with the BASIC comparison operator "<":

```
    30 IF X < 9 THEN GOTO 50
```

and in assembler:

```
        CPI  9
        JC   XXX
```

In BASIC, it is easy to make more complex comparisons, such as:

```
    30 IF X < = 9 THEN GOTO 50
```

To "translate" this directly into assembler, requires:

```
        CPI  9
        JC   XXX
        JZ   XXX
```

Why? Because simply, there is no direct translation for the "less than or equal to" operator. A simple alternative would be to change the technique, but not the logic:

```
        CPI  10
        JC   XXX
```

since "less than 10" is the same as "less than or equal to 9".

A second thought, combines the fact that carry is set on a "less than" comparison, with the fact that carry is a useful bit to use to indicate success or failure of some subroutine. A typical subroutine might be one to test if the accumulator contains an ASCII printable digit, 0 to 9, returning carry if the digit is not. I'll have to introduce one trivial instruction to do so:

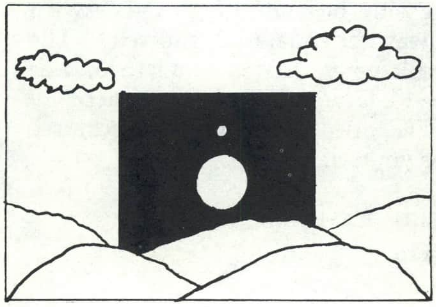
CMC

CMC means "complement carry" and does nothing more or less than that. Here is a use:

```
;
;ROUTINE TO TEST IF THE ACC
;CONTAINS AN ASCII DIGIT,
;RETURNING CARRY IF NOT.
;
CK09 CPI   '0'    ;<'0' SETS CARRY
     RC           ;RET IF < '0'
     CPI   '9'+1  ;'9' OR LESS
                  ;  SETS CARRY
     CMC          ;FLIP CARRY
     RET          ;RETURN, WITH
                  ;  CARRY SET IF
                  ;   '0'-'9'.
```

This example shows very well why I like assembler programming. The routine takes 7 bytes, and executes, on a 4MHz system, in about 3 to 7 millionths of a second (depending upon whether the RC is executed, or control passes on to the last 3 instructions.

That wraps up another part of the tutorial. Like to keep score? There are 7 instructions to go, in the "Input/Output and "Other Instructions" sections of the tutorial. Then, the last three sections of the tutorial: "PITFALLS", "SUBROUTINES" and "CP/M INTERFACE".



Correction

(Editor's Note: This note refers to the review of FABS and MAGSAM which appeared in the December issue, Vol. II, No.7 pp15-17.)

Greg Scott of Micro Applications has informed me of some errors in the article on MAGSAM and FABS. MAGSAM does not use a hashing structure but instead maintains the primary index in ASCII sorting sequence and uses a binary search to locate keys. The structure in figure 2 of the article is still a good depiction, however. (The figure references are reversed in the article.) The use of an overflow bucket still can slow access to keys down, but the binary search is probably a little faster than a hashed search. I made a statement that MAGSAM requires you to stop and use a special program to reuse deleted space. There are delete functions which do make space from removed keys and records immediately available. The reorganization referred to is useful for records which are logically deleted but are not removed immediately.

In my discussion of FABS, the figure really only represents nodes of two keys. There is actually a node at the next level associated with each key in the node. That is, the pointer to the next level node is part of each key in a node.

Steve Patchen

Notice

This issue (dated February 1982) was mailed from New York on January 27th. We would appreciate your help in tracking the deliveries. Should you have any comments, or any problem with the timeliness of this issue, please call our Subscription Department at (212) 722-1700, or write to: Lifelines Subscription Department, 1651 Third Ave., New York, N.Y. 10028.

Zoso

SOFTWARE PIRACY ETC.

Some six months ago, I chanced upon the June, 1975 issue of '73' (a ham radio magazine). The idea for what follows was born then, but until I saw the December '81 Kilobaud/Microcomputing and read Publisher Wayne Green's editorial on 'Program Theft', I held off. I'm not holding off any longer!

In his December editorial, Mr. Green goes on the warpath against the rapacious scoundrels who have been ripping off Instant Software products. Mr. Green's pique is ostensibly backed by a $10000 reward for; "...information which enables us to get a conviction of someone copying our copyrighted programs..."; is this phrasing as clumsy as it sounds? I don't think so! Let's wait a year and see if Mr. Green and his ten grand have parted company.

Mr. Green goes on to say that this pretax bounty is sufficient to take two people on an all-expense paid trip around the world. Maybe so, but only if those lucky travellers are willing to eat lots of wild berries picked during very long walks or whatever they might find floating during some even longer swims. Oh yes, back to the thieves; amongst other ideas for nailing them, Mr. Green suggests collecting hard evidence with a concealed recorder. Sounds like a job for Special Agent Maxwell Smart... Mr. Green volunteers that he is never without his microrecorder. Caveat locutor!

If nothing else, one would have to assume that thievery is one thing Mr. Green will not tolerate. Or would one? The June, 1975 issue of '73' magazine was published by none other than Wayne Green and, beginning on page 67, is the final installment of an article by Spenser Whipple Jr. entitled 'Inside Ma Bell', which offers little more than detailed instructions (complete with schematics) for building Red boxes and Blue boxes. Red boxes are used to trick older pay phones into 'thinking' that coins have been inserted when the phone is being fed nothing more than air (in the form of coded tones). Blue boxes enable one to make long distance calls for no charge. The Blue boxes described by Mr. Whipple were based on a pair of Intersil 8038CC devices, and if the larcenous 'hobbyists' of 1975 didn't at least improve on the published design by using high quality multi-turn trim pots, I wouldn't be surprised to hear that a few of them soldered their way straight into some Federal work farm.

So what's the real story, Mr. Green? Is Ma Bell fairer pickings than Instant Software; or is it simply that you assumed we'd all forgotten? In light of your latest moralizing, I'd wager you'd rather not cope with a sudden deluge of requests for reprints. Separately, I'll bet you don't want the Telephone Company associating your good name with a sudden theft of service epidemic.

N.B. - As of June, '75, one could contact either Mr. Whipple or Mr. Green c/o '73' Magazine - Peterborough, NH 03458. At least one of these gentlemen may still be reached at that address.

I have not seen GIGO's column in InfoWorld again. I hope it wasn't anything I said last time. You know, maybe I was too hard on her. I mention this because, as it turns out, she was pretty much right on target about 'The Last One'. I have not yet seen 'The Last One' generate a program, but during the last few weeks, I have seen it completely fail to do so on a number of occasions. This may sound discouraging, but don't abandon hope just yet. The professionally written copyright and ownership notice which accompanies 'The Last One' (Version 1.0) implies a product which someone, someday may well want to own for free. Beware pirates; read that notice carefully; on first glimpse, it would seem that there is a standing $5000 bounty on your heads. Not so! The owners of the copyright merely "reserve the right" to pay this sum for information resulting in conviction of anyone who distributes unlicensed copies. That's quite some way to set the stage for an honorable business relationship, don't you think?

How does the old saying go? "Money saved is money earned". It seems one can 'earn' a tidy sum by 'reserving a right'. What the deuce; I'm entitled to make a living too! I think I'll just 'reserve the right' to help myself at Fort Knox.

Some 'must' reading on the subject of software piracy can be found in the January, '82 issues of BYTE and Esquire. The Esquire article profiles some individuals who have proven especially adept at defeating various protection schemes for Apple software and the article identifies a surprisingly big name in the computer industry whose skills were honed as a 'Phone Phreak' (the unofficial job title of a person who rips off the Phone Company). Don't miss reading this! The same applies to Jerry Pournelle's piece in BYTE. I can't recall having read a more intelligent analysis of the software piracy problem.

Sometimes I hammer away too frequently on topics which I find especially troublesome. Thanks to the good folks at BYTE, I'll be able to keep this failing in check (at least this time). Read the letters on page 18 of the January, '82 issue and you'll know just what I mean.

There is a computer store in New York City which has really found a way to overburden a microcomputer while overburdening their customers at the same time. They have implemented a replacement for the common cash register. It works like this: Every transaction (even buying a single magazine) is keyed into their computer, complete with unwieldy product codes and a plain English description. As you [have to] wait, their computer is updating on-hand cash and inventory files and Lord knows what else. After far too long, you do get a nice detailed

(continued next page)

receipt, printed before your very eyes. Owing largely to the obstinacy of [at least some of] the people who ring up the sales there, (perhaps carillon would be a better word than 'ring'), you won't see any change or merchandise until this little Passion Play has run its tiresome course.

One of New York's most prestigious department stores does something equally odious on a much larger scale. The offenders here are the many point-of-sale terminals attached to a time sharing machine. Whenever you buy something, all the gibberish on the price tags must be keyed in exactly. Only when the main computer is in a cooperative mood, (which is not always), will a transaction proceed without substantial delay. From what I've seen, loyal customers (and there are many) probably sacrifice many months of their lifetimes to this system alone.

I prefer shopping the old fashioned way. I like simple register receipts, devoid of product codes and I like getting them pronto. How a business maintains its aftersale records is neither my concern nor my responsibility and I deeply resent this work being done while I wait. One of these days, maybe enough customers will get so fed up with this kind of inconsiderate excess that certain businesses will be obliged to buy much more powerful computers, rediscover service bureaus or simply relearn how to do it all by hand.

Now there's no reason for me to single out the stores in question, so I won't. Okay, just a couple of hints... If you walked to the computer store, starting at the Empire State Building, you'd have to walk less than four blocks. The department store sits atop a busy subway station, less than two blocks from a bridge to Queens.

I think it's safe to assume that there are fortunes to be made by writing quality applications for the IBM Personal Computer. I'm hoping to grab a piece of this action myself. Normally, my first step would be to buy one of these machines to use as a development system. Unfortunately, it's not quite so simple. I'm only geared up for 8″ drives and IBM doesn't presently offer them. If forced to purchase a machine which uses the smaller 5″ units, I'll not only be stepping backwards so to speak, but I'll be turning my back on many dollars worth of 8″ hardware, media and know-how which are already on tap. This dilemma of mine will cease to be a factor when some enterprising hardware manufacturer comes up with a good 8″ DMA disk controller for the IBM P.C. An even more efficient solution would be a terminal which can emulate the IBM unit, graphics and all. Won't someone help? Please!

OLD CONTEST NEWS

Due to an incredible work overload, I had to farm out the judging duties for the first contest (see *Lifelines* - October '81). Naturally, as promised, the lucky winners will still be awarded their prizes.

BROKEN PROMISES

Last time, with the best of intentions, I offered a preview of the second contest. As originally envisioned, that contest would have tested your understanding of a rather complex, largely uncommented BASIC program. There has been a change of plans here too. In truth, the program was finished and tested early in December. Sadly, it was almost four pages long, which is simply too much space to devote to a contest problem. I'm new at the contest game, so I hope you'll understand. Actually, it's better this way; the [second] new contest involves only three words and presents a more interesting challenge in the bargain.

THE SECOND CONTEST

ORANGE - PURPLE - SILVER ←— The 'Key Words'

There are many things which can accurately describe the 'key words'. For example; each is a color, each is six letters long, each has two syllables, four letters in the middle of each word can be used to form other words: RANG-PURL-VILE, and so on. If you look at the 'key words' long enough, and especially if you let your computer help you look, you will find dozens (or possibly hundreds) of equally valid answers. However, from what I've seen so far, one common attribute is more remarkable and unusual than any of the others. This 'special answer' presently resides in a sealed envelope, safely hidden somewhere in *Lifelines'* office. I am the only person who knows what the envelope contains, and that envelope will be only be opened in the presence of the contest judges the day after the contest deadline.

OBJECTIVE AND PRIZES: The object is to submit the longest valid list of things or qualities common to the 'key words'. Each item in the list will be worth one point, The 'special answer' is worth twenty-five points. The highest scores win. The prizes are essentially as described for the first contest; see *Lifelines* - October,'81.

CONTEST RULES: Except as noted, the rules are the same as for the first contest.

Entries for this contest must be postmarked no later than April 1, 1982. Number your answers and indicate at the top of the [first] page how many valid answers appear in your entry.

Answers duplicated semantically or otherwise in a single entry list may be disqualifying. For example; if you noted that the three 'key words' were all adjectives and later noted that all three were 'parts of speech', your entry might be discarded regardless of other redeeming merit.

Patently invented guesses will disqualify an entire entry. This rule is intended to eliminate answers which can't readily be validated. For example; you might happen to know that the key words accurately describe the most popular eyeliner shades worn by trendies in Albanian discos. Whether true or not, such esoterica will be disqualifying unless fully documented. If the judges deem that an otherwise strong entry suffers only from a single, marginal violation of this rule, that answer rather than the entire entry will be rejected.

HELPFUL HINTS: If you have any doubts as to whether an answer in your list will be judged acceptable, don't include it.

About the 'Special Answer': By my rough estimate, this should appear in about half of the entries. Your computer will be of no help whatever in getting this far; rely on common sense and your imagination instead. I doubt that PHDs will have any measurable edge over bright sixth graders. In fact, familiarity with the English language (both spoken and written) at an advanced sixth grade level is about all you'll need. This applies regardless of regional dialect or patois. The 'special answer' is not facetious; jokes and trickery are not involved.

About the other answers: Here, intelligent use of your computer may well afford you the winning edge.

IN CLOSING

Finally, a few questions from the mailbag and some answers from the heart:

Q: Are you English or American?
Z: Smile when you ask me that, Varmint.

Q: What do you look like?
Z: Pretty much like my passport photo. I'm 5′11″ when I slouch and I've got big brown eyes.

Q: What do you do for fun?
Z: Smile when you ask me that, Vixen.

Q: Do you write under another name?
Z: Yes, 'Zoso'.

Q: What do you do with computers?
Z: I use them.

Q: I'm thinking of buying a computer. What should I get?
Z: Enough money for your dream machine with enough left over to cover the divorce which often follows.

Well guys and gals, that's it for this time. I'll be coming back at you soon. I'm counting on some good contest entries this time, so get cracking. To help get you in a contest mood, I buried a truly dreadful pun somewhere before the Contest News. See if you can find it.

Take Care,
Zoso

Letters

December 10, 1981

Dear Editor,

I read your short report about a supposed bug in the Z 80 CPU which was discovered by a Mr. Robert Burns.

I would suggest that this is a bug in the Zilog documentation and not in the Z80 CPU. If you look in the Mostek Z80 Programming Manual Version 2.0 from October 1978 you will see that the carry flag is listed as "unknown" on these block I/O instructions, as its state is dependent on the data input or output.

We would be grateful if you would bring this to the attention of your readers in order to correct the impression that there are unknown faults in the Z80 CPU.

Best regards

W. Finkle
MOSTEK GmbH

December 4, 1981

Dear Kelly:

I'm sorry it's taken me so long to get back to you. The last month has been a hectic one, to say the least.

Concerning your request for schematics and source code: we will be selling a technical reference manual early next year through our dealer network. I'm not sure what revision level board you have in your machine, so you'll need to supply me with that information (your serial number will be fine) before I can proceed with your request.

I'd like to make a few comments about your article, also.

1. You have an old version SYSGEN. You should have received a letter explaining our updating policy for new keyboard, ROM and software. The problem goes away with the fix.

2. If you liked the documentation in round one, you'll love it come January 1st when we begin supplying a new manual with a completely rewritten tutorial on ALL of the software we provide. I guess I have the usual writer's ego, but I think that this will be one of the best user manuals around (if the comments on my CP/M book and my editing of InfoWorld are any indication, I might even be right!)

3. Your comments concerning free updates are amusing. Having been on both sides of the fence, all I can say is that Osborne Computer Corporation's policy is to fix the things that we should have done correctly. On the other hand, when significant changes in product occur (double density, WordStar version 3.0, etc.), we will have to charge for these items.

4. I would be careful about the 5″ winchester ideas. We would offer such a goodie if it were practical, but no one can show us working models which will sustain the amount of abuse a portable machine gets. If it's hard disk you're after, Corvus and others already have implementations that run off the IEEE-488 port, and I'm quite impressed with the speed (faster than the same machine on an Apple with a DMA card).

5. The reason we didn't explain about the diagnostics is that a) they might not continue to stay in ROM, and b) the messages aren't extremely enlightening to novice users. You should warn folks that using the disk test erases any and all data on a diskette, so don't use your data diskettes to perform a disk test. Also, the disk test performs no retries. In other words, if you get a few errors, nothing to worry about; if you get gobs of errors, you probably do have something wrong.

6. We will NOT make the BIOS configurable by the user. With the update you'll find that we've made it so most everything you'd want to send information to has been implemented in BIOS. If you know enough to reconfigure a BIOS, you know enough to create your own from scratch. We do not want to get into a situation where we have to support multiple versions of the BIOS or explain how to change things. I will caution you that changing your BIOS also may mess you up for some nifty new offerings when double density becomes available in February (like an Osborne which can

(continued on page 26)

Macros of the Month

Edited by Mike Olfe

(*Editor's Note*: Congratulations to Ward Christensen, who, as our winner, has graciously prolonged the existence of this column via his contribution below. If only all those out there hoarding PMATE macros would be so kind! Send in those macros—Mike Olfe.)

DECIMAL NUMBER TO @1

The following macro will scan a file until it encounters a decimal digit. It will then "eat" all subsequent decimal digits (remove them from the file), leaving the decimal value in variable 1. You can then typically, do arithmetic on it, or move it to the end of the line, or whatever, then insert it back in the file with the (variable 1, inserted as an ASCII number). This is my "permanent macro 'D'":

```
^XD[@t=0[%]((@t>47)&(@t<58)){_}{m}]0vl
[((@t>47)&(@t<58)){@1*10+@t-48̄vld^}{_}]
```

which means:

```
^XD                ;permanent macro named "D"

[                  ;outer loop
@t=0               ;if at end of file
        [%]        ;exit macro
((@t>47)&(@t<58))  ;is char under cursor a decimal dig?
        {_}        ;yes, exit to following "["
        {m}        ;otherwise skip this character
]                  ;and loop back
;
;found a decimal digit
;
0vl                ;init var 0 to a 0
[                  ;loop "for each digit"
((@t>47)&(@t<58))  ;if it is a digit
        {          ;then
        @1*10      ;multiply previous var 1 by 10,
        +@t        ;add current ascii value of digit
        -48        ;make in binary ('0' = decimal 48)
        vl         ;place back in variable 1
        d          ;delete the digit processed
        ^          ;loop "for each digit"
        }          ;end of "then"
        {          ;else
        _          ;exit, no more digits, value is in @1
        }          ;end of "else"
]                  ;end of loop "for each digit"
```

Here is an example of the usage of my .D macro. For example, if I have a series of labels:

RTN5 blah blah

RTN6 blah blah

and I want to insert a new routine after routine 5, then"renumber" the 9 labels after it, I manually insert the new label, the position to the line above RTN6, the first one to be renumbered one higher, and type:

```
9[s
RTN$.d@l+l\]
```

which means:

```
9[              ;repeat 9 times
s
RTN$            ;search for (CR)RTN
@l+l            ;add l to it
\               ;insert number back into the file
]               ;and repeat the 9 times.
```

FORMAT FILE NAMES IN COLUMNS

Mike Olfe showed a way to format a series of file names (inserted via xlfilename.filtype) in columns, by setting tab stops. Here is an alternative way to format names, not as sophisticated (for example, it doesn't know when to quit), but it also doesn't change the tab stops. Also, I like to call it my permanent "F" macro, but sometimes I accidentally type ".f" when I meant just the "f" (format mode), so this macro prompts "ready" and you must hit C/R to start it:

```
^XFgready$@k=13'[%] [[1@t=0[%]-m@x>64{m^}_] [(@x&15){9i^}{_}]dqr]
```

which means:

```
^XF             ;my "F" permanent macro
gready$         ;prompt with "ready"
@k=13'          ;if key pressed is not C/R (' = "not")
        [%]     ;then abort the macro
        [       ;"else"
[               ;loop for one line
1               ;go down 1 line
@t=0            ;if at end of file,
        [%]     ;       exit macro, all done
-m              ;move back over a C/R
@x>64           ;if the column is > 64
        {m^}    ;       then skip C/R, loop
                ;and exit
]               ;end of loop for one line

[               ;loop inserting tabs to align
(@x&15)         ;if in a tab, multiple of 16,
        {9i^}   ;       then insert tab, go
                ;       back to "["
        {_}     ;otherwise exit, we are at tab stop
]               ;end of tab loop
d               ;delete the C/R
qr              ;display
]               ;and loop for next name
```

GO TO A SPECIFIC LINE NUMBER

Here is one of my "favorite" macros, because it it so simple. "89.g" will go to line 89 of the file. Since PMATE shows the current line all the time, it is useful to remember a particular place you want to be, and use .G to go to it. A particularly good general use of .g is to resume editing a file where you left off, after doing a "xj" save command. Since PMATE puts the current

(continued next page)

line number in the upper right corner, you look at it, and, for example if it was 1095, type "xj$1095.g". This will save the file, (the "$" being an escape, so 1095 is not taken as the filename), then when editing resumes, go to line 1095 and continue. Here's the .g macro:

↑XG@a-@ll

which means:

```
^XG                 ;permanent macro named "G"
@a                  ;get the requested line number from "nn.g"
-@l                 ;subtract the current line number
l                   ;and move than many lines.
```

HEX NUMBER TO @1

Just as .D put a decimal number into @1, .H puts a hex number into @1. This can be used, for example, to change a value in a file, from hex to decimal: ".h@1/".

```
^XH[((@t>47)&(@t<58)){_}{m}]0vl
[((@t>47)&(@t<58)){@l*16+@t-48vld^}{((@t>"@)&(@T<"G)){@l*16+@T-55VlD^}{_}}]
@t="H[d]@t="h[d]
```

which means:

```
^XH                 ;perm macro named "H"
[                   ;repeat search for leading decimal digit
((@t>47)&(@t<58))   ;if a DECIMAL (leading) digit:
        {_}         ;       then break out of macro
        {m}         ;       otherwise move to next digit
]                   ;and loop for leading digit

0vl                 ;init var 0 to 0
[                   ;repeat "for the number"
((@t>47)&(@t<58))   ;if decimal
        {           ;then
        @l*16       ;       multiply prev tot by 16
        +@t-48      ;       add in this digit minus ASCII offset
        vl          ;       put it back in vl
        d^}         ;       delete digit, and loop

        {           ;"else"
        ((@t>"@)&(@T<"G)) ;if it is a hex digit
                {       ;then
                @l*16   ;multiply prev tot by 16
                +@T-55  ;add in this digit minus ASCII offset
                vl      ;put it back in vl
                d^}     ;delete character, and loop

                {_}     ;else exit past next "]"
        }               ;end of digit loop
]                       ;end of loop

@t="H                   ;if there is an "H" (as in 5CH)
        [d]             ;then delete it
@t="h                   ;if there is an "h" (as in 5ch)
        [d]             ;then delete it
```

SCROLL THE DISPLAY

At times, I would like the display to just scroll slowly, without having to type any scrolling keys over and over. This macro scrolls until the break key character is pressed:

```
^XS[lqdqr]
```

which means:

```
^XS             ;Permanent macro named "S"
[               ;loop
l               ;go down a line
qd              ;delay
qr              ;display
]               ;and repeat
```

Ward Christensen

Software Product Brief: DDUMP and DTEST

Jim Mills

These two utility programs are from Elektrokonsult, a firm in Norway. To borrow their product descriptions from their ad sheets, "DDUMP (Disk Dump) and DTEST (Disk Test) are two advanced disk utilities which work with most types of drives and disks used with CP/M, including 5" and 8", hard and soft sectored, single and double density, single and dual sided disks. Both programs automatically adapt to the actual number of tracks and sectors per track for most types of diskettes without manual intervention. Both programs run on any 8080, 8085 or Z80 system." Well, that sounds fairly straightforward. They probably use the CP/M version 2 disk allocation tables in the BIOS to do all the disk I/O, so they adapt to any disk format because there is no adaptation necessary. Ok, what do these programs do for you?

I looked at DDUMP first. Its description reads: "DDUMP is a sector oriented disk dump utility, which makes it possible to examine and modify any byte(s) on any sector, addressed by track and sector number, OR addressed by allocation block (or 'group') number. Sector contents are printed and/or displayed — and may be patched — in both hex and ASCII formats. DDUMP allows you to dump and patch data not otherwise accessible by CP/M, and it may for example be used to examine test files for control characters, tabs, etc, study how CP/M allocates disk storage, examine and repair a damaged disk and recover deleted files and lost data." The first thing I thought of when I read this was Ward Christensen's DU (Disk Utility) program, which does all of the above and is in the public domain (free). Since I am somewhat familiar with DU, I set out to compare the two programs. Maybe DDUMP has some features not in DU, or vice-versa.

The first thing to do was look at the instruction sheets (user's guide) that accompany the disk. This 16 page document is well written and easy for most any computer owner to understand, assuming a minimum knowledge of how CP/M and disk drives work. The user's guide starts right out with a summary of commands followed by a "first time through" section for the new user of DDUMP. Following this is a large section elaborating upon each command, then a section on error messages, followed by several pages of discussion on how disks work, how CP/M allocates disk space, hints and kinks, how to patch DDUMP for uppercase printout only, and references to two books and the CP/M system documentation from Digital Research. My first impression, without actually running the program, was that it was not as sophisticated as Ward's DU, but I decided to keep an open mind, so I ran DDUMP.

It came up and ran with no problem, and it did everything the user's guide says it should. I did a dump of my system tracks, the directory, and a short program out on disk. Their sector 'dump' format is nice in that it gives a 'ruler line' at the top of the sector dump, indicating the 'ID' or each byte in the dump (ie: address 3F within the sector). I was not impressed with their sector patching mode (similar to the 'S' command in DDT) for either hex or ASCII patching. I did notice, when dumping the directory, that R/O and SYS files showed up as "..M" in the ASCII part of the dump (ie: MAC.COM listed as "MAC..M", the high bits of the 'C' and 'O' in COM having been set to ones by CP/M when I stat's the MAC.COM file to R/O and SYS.) DU is smart enough to know that these are still .COM files. All in all, I was underwhelmed, having used Ward's DU for some time now. There is no fair comparison, DDUMP is outclassed by DU, which is available from the CP/M Users' Group .

Well, having finished with DDUMP in less than two hours, including the

(continued next page)

writing of this portion of this article, I turned my attentions to DTEST. Here's Elektrokonsult's blurb: "DTEST is a fast test utility which tests a disk by writing and reading a test pattern on every sector. Disk errors are listed on the printer and/or console, with track-, sector and allocation block (or 'group') number. All bad sectors are locked out from later use by CP/M by automatically collecting them in a write protected 'garbage' system file. The program also tells you whether a particular bad sector is on the reserved system tracks, on the disk directory area or on a sector which will never be used by CP/M. DTEST may also be used to test disk drives. Save money and frustration by employing DTEST to test every diskette before you start using it with CP/M. By doing this, you will reduce the probability of later disk crashes, and you may even start using those 'not 100% perfect' disks which you get now and then."

As with DDUMP, I start with the documentation. This time it's a 13 page document in a style similar to the DDUMP user's guide. There isn't as much to tell as with DDUMP, the program only has a few options: it asks you if you really want to wipe all data on the disk, if you want a quick "read only" test (or write/read test), how many passes over the disk you want (maximum of 99), whether or not to output bad sector information to the printer, and which drive to use. It then asks you to insert a disk and type return to start the test. "During the test, DTEST will report all bad sectors to the printer and/or console, with corresponding track and sector numbers, and to which allocation blocks the bad sectors belong." (from the user's guide.) Well, I guess it's time to try it out.

I first used the "read test only" option and immediately got some errors. All that DTEST tells you is the track and sector numbers, and the block (or group) allocation numbers (if appropriate) for the bad sector. It does not tell you the data in error. In fact, I found that DTEST reports false errors if you do a quick "read test only" if any single data byte on the disk is not 'E5' hex. That seems a little inconvenient, unless you've just finished formatting your disk. If you do the longer write/read test, then data is written to disk as all E5's, so there is no problem reading it back. On my single sided single density system (Tarbell controller, 8080 CPU) the read-only test ran for one and one-half minutes. The longer write/read test ran for two minutes and forty seconds.

When DTEST detects an error (or errors), it checks to see if they are in the system tracks or the directory or the data area. If the error is in the system tracks, DTEST warns you not to use the disk as a system disk. If the error is in the directory, it tells you to abandon the disk. However, if the error is in the data area DTEST allocates the bad sector(s) to a read-only system file named BDSCTRS.GRB, which I verified. That's about it.

So, how does this compare to similar programs, such as FINDBAD from CP/M Users Group volume 20? Knowing how the programs function, I think they are probably both about the same functionally, although DTEST has a few more options built into it. Both programs create a "bad sector" file that is reserved under some strange filename, such as [UNUSED].BAD (FINDBAD uses that one). If you STAT these to system files, which DTEST does automatically, then PIP won't read them, so you can still do "PIP A:=B:*.*" but you can't do a full disk copy, such as with DFOCO or COPYFAST, without having bad sector errors, as these programs copy track by track, rather than file by file. You could, I suppose, alter your track by track copy program to search the directory for a particular filename, look at which groups (or blocks) are allocated to that filename, and *not* copy the sectors allocated to that group, or groups. But, I digress.

Elektrokonsult sells these two programs for $29.95 each and they are available on either standard 8-inch SSSD diskette or on Heath 5-inch CP/M (no mention as to whether it's hard or soft sector format for the Heath). I, personally, would not spend the money for either program, mostly because equivalent programs are available from the Users Group, but that is strictly my opinion. Others may find these utilities to be just what they want, although it would surprise me.

What really does surprise me, and has for some time, is that I keep seeing various CP/M utility programs being offered to the public at prices from twenty dollars up, when there are similar and/or functionally identical programs available for the cost of media (and mailing and copying charges that total $8 per disk for the CPMUG disks) from the CP/M Users Group, or SIG/M in New Jersey. It's relatively free software, and quite a bit of good stuff, too. Yet I am constantly amazed to see for sale utilities that duplicate programs (easily available programs) that are in the public domain. These people are selling (and presumably writing) CP/M programs. This usually requires a fair knowledge of the operation of CP/M and a familiarity with the CP/M marketplace, doesn't it? Well, enough complaining. I would like to offer some constructive criticism for those who write and sell CP/M system utilities. I would recommend that all such programmers acquire the entire CP/M Users Group and/or SIG/M libraries and familiarize themselves with the programs found therein. It would save end users, product reviewers, and others (I'm sure) a lot of grief over purchasing a product one week, then finding its equivalent (for free) in the public domain the next week. I shall get down off of my soapbox, now. Thank you for reading.

(continued from page 21)

read and write IBM, TRS80, Xerox and Zenith diskettes). I guess our position is best summarized as follows: if you want to fiddle around with the system, we'll provide enough information so you can, but you're on your own; if there's something that should be in the BIOS and isn't, we'll put it in and support it.

I hope this clarifies some of the points you raise in your article. If you need any further information, please don't hesitate to call me, I promise I won't take so long to get back to you next time. Also, if you still want schematics, forward your serial number to me, we have six revisions of the motherboard now, and it would be nice if your schematics matched.

Sincerely,

Thom Hogan
Director Software &
Publications
Osborne Computer
Corporation

Using the CP/M-80 BIOS For Direct Disk Accessing

Ron Fowler

Want to write a disk utility that will run with anyone's disk controller? Maybe a copy program that'll work with single *and* double density diskettes? Maybe even hard disks? These kinds of programs are generally written to work on one specific disk controller, and are not usually transportable between unlike systems. It is possible, however, to write these programs in such a way that they will automatically adapt themselves to whatever disk environment they execute in, by using the techniques I'll describe in this article.

I've used these methods to upgrade three very fine public-domain utilities(1) so they can perform this "dynamic" adaptation. Previously, each of these utilities contained a long series of conditional statements for different hardware configurations. If yours was not on the list, you either couldn't use the utility, or had to spend a considerable length of time determining the parameters to "plug in". Further, if your hardware *was* included within the conditionals, the utility had to be separately assembled for each different disk format to be used. Clearly, this nuisance limited the use of otherwise very valuable programs.

In their present form, these utilities need not even be reassembled by the user. They run on virtually every disk controller/CBIOS combination running CP/M-80 2.x, and *most* systems running CP/M-80 1.4. And, best of all, they run correctly when intermixed formats of disks are encountered; this means you can, for example, run the SAP utility from a double-density disk on drive A, and expect it to work correctly on a single-density disk on drive B.

The key to understanding these disk-accessing techniques is the fact that CP/M-80 proper (i.e., the BDOS) runs unchanged among the many disk configurations to which it is interfaced. The *only* portion of CP/M-80 that changes from system to system is the CBIOS (except for versions of CP/M-80 prior to 2.0; I'll explain later how to "work around" CP/M-80 1.4). It follows, then, that by accessing the CBIOS in much the same way that CP/M-80 itself does, transient programs may also adapt to different systems.

There is one mechanism defined within the CBIOS that determines the physical disk characteristics of any given drive; this mechanism is called the Disk Parameter Block (DPB).

DPB OVERVIEW

The DPB is used by CP/M-80 itself to determine the structure of the disk system to which it is interfaced. In CP/M-80 versions prior to 2.0, the DPB was part of the operating system itself, and information regarding its location and structure was available in general only to OEMs (Original Equipment Manufacturers). The DPB had to be patched to accommodate non-standard disk systems (i.e., non eight-inch floppies). Since most of the parameters were byte values rather than word values, early versions of CP/M-80 were rather limited in the disk capacity they could address.

CP/M-80 2.0 changed all that, by moving the DPB into the BIOS (Basic I/O System), leaving the DPB definitions to the system implementor. This allowed simpler interfacing to various disk formats. In addition, most of the restrictions on disk capacity were removed by changing the byte values to word values, allowing large-capacity hard disks to be directly addressed by the operating system (CP/M-80 1.4 *could* be interfaced to hard disks; this was usually done by segmenting the disk into "regions", only one of which could be active at any one time. A utility program was usually provided to activate/de-activate these regions. This was a nuisance to both the user and the system implementor).

The specific format of the DPB is detailed in Table 1 for both CP/M-80 1.4 and 2.0, and is physically ordered as shown.

The parameters are defined as follows(2):

SPT — This is the number of 128-byte sectors per track, increased from a byte value in CP/M-80 1.4 to a word value in CP/M-80 2.0. this value should not be confused with the *physical* number of sectors per track; if your system has ten 512-byte sectors per track, then this value will be 40.

BLKSHF and BLKMSK — These parameters are used to determine CP/M-80's block size (also called "groups" by some people). A "block" is the minimum amount of disk space that can be allocated by CP/M-80, and is always a group of sectors that is an integral multiple of 8. Specifically, BLKSHF is defined as the logarithm of the number of sectors per block to the base 2 (LOG2 (SPB)), and BLKMSK is SPB-1. The reason for this rather cryptic way of specifying block size is that this form provides the fastest way of doing disk arithmetic, using shifts and logical operations on the parameters themselves. BLKSHF and BLKMSK can take on values as defined in table 2.

DSM - This is the maximum block number available on the disk, with block numbering starting at zero.

EXTMSK- this parameter is present in CP/M-80 2.0 only, and is used to specify the number of extents contained in a directory entry (an "extent" is a 16K group of data used by CP/M-80 to partition files). Under CP/M-80 1.4, one directory entry described one and only one extent. CP/M-80 2.0 allows up to 16 extents to be described by one directory entry, under control of this parameter. It should be noted that, although the ALTERATION GUIDE seems to indicate that EXTMSK must be non-zero for certain disk characteristics, this byte should always be zero for random-access compatibility with CP/M-80 1.4 programs. The following table is from the ALTERATION GUIDE, and is present also in Digital Research's MP/M II SYSTEM GUIDE, as a MAXIMUM value (only) for EXTMSK:

(continued next page)

BLOCK SIZE	DSM<256	DSM>255
1024	0	N/A
2048	1	0
4096	3	1
8192	7	3
16384	15	7

DIRMAX - This parameter determines the total number of directory entries which can be stored on a drive. It is defined as the maximum directory entry number, with numbering starting at zero. Thus, if there is room for 128 directory entries, this value would be 127.

DIRALO, DIRAL0, DIRAL1 - These bytes are used to allocate space for the directory. DIRALO is the entire parameter (8 bits) used under CP/M-80 1.4; DIRAL0 and DIRAL1 combined form the 16-bit value used under CP/M-80 2.0 (DIRAL0 is the most-significant byte, DIRAL1 is least). Each bit in this parameter reserves *one* block of data for the directory. It should be noted that this parameter in no way determines the directory size; it merely serves to allocate space in CP/M-80's allocation bitmap, thus preventing any file allocation.

CHKSIZ - This is a value present only under CP/M-80 2.0, and specifies the number of directory sectors to perform a checksum on. This checksum is used by CP/M-80 to detect changed media; when using non-removable media (such as hard disks), this value can be set to zero, thus specifying *no* and allowing a much faster disk log-in. With removable media, CHKSIZ is defined as (DIRMAX+1)/4, which is simply the number of sectors occupied by the directory.

RESTRK - This value is the number of reserved tracks (commonly called the "System Tracks"). These tracks are not accessible by the file system for directory or data allocation, and usually contain the operating system image, and a bootloader to load the system. Note that when large physical disks are segmented into smaller logical disks, this parameter will contain *all* of the tracks of any previous logical drives.

SECTOR TRANSLATION

Also associated with direct disk accessing is the concept of sector translation. This is a technique used to associate the physical disk sectors with the logical sectors requested by CP/M-80. Consider the physical movement of the read/-write head over the diskette; if sectors are numbered consecutively, then as one sector is read, it is very likely that, as the sector just read is being processed, the head will move past the next sector or two. This would require waiting for the entire disk to make another revolution before reading the next sector, effectively limiting the access rate to one sector per disk revolution.

Sector translation allows more efficient disk accessing by ordering the physical sectors in a "skewed" fashion. On any given track, we will write the first sector, skip a number of sectors, then write the second sector, skip again, and continue this pattern for the entire revolution of the disk. On the next revolution, we write some of the sectors skipped on the first revolution, and continue with this until the entire track has been written. This is accomplished by translating the "logical" sector number requested by CP/M-80 to a physical sector number using a table called the Sector Translation Table.

Whenever CP/M-80 requests a sector (via a SETSEC call on the BIOS), it first calls a BIOS entry point called SECTRAN, which performs this translation using the sector translation table. SECTRAN will usually use the passed logical sector number (with sector numbering always starting at zero) to calculate an offset into the sector translation table to extract the physical sector number.

LOCATING THE BIOS VECTOR

The BIOS entry points are accessible to a transient program by using the jump at location 0 (the "warm-boot" system entry point) as a pointer into the BIOS jump table. The location of the base of the table can be found by executing the following code sequence:

```
LHLD   CPBASE+1;get address field of jump at loc 0
DCX    H       ;it points to second jump instruction
DCX    H       ;so decrementing pointer to the first
DCX    H       ;jump instr. yields table start
```

Note that CPBASE (equated to 0 in "standard" CP/M-80 systems) is the "base" of CP/M-80; I always define CP/M-80 locations using this variable in order to facilitate moving them to the special versions of CP/M-80 created for systems (such as the TRS-80 model I and the Heath H8) which have ROM in low memory. Such systems have CPBASE equal to 4200H.

I've found the best way to access the BIOS is to move the entire entry jump table down to a local area, and perform calls on the local table whenever it is necessary to do BIOS calls. This is accomplished by the following subroutine:

```
GETVEC: LHLD    CPBASE+1;get BIOS table pointer
        DCX     H       ;as above
        DCX     H
        DCX     H
        LXI     D,MYTABL;get pointer to local table
        MVI     B,3*13  ;13 entry pnts, 3 bytes each
MOVE:   MOV     A,M     ;perform the move
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     MOVE
```

Before this subroutine returns, it should test for CP/M-80 version number (because the sector translation routine, SECTRAN, is not provided in the BIOS jump table under CP/M-80 1.4):

```
MVI    C,12     ;"GET VERSION" FUNCTION NUMBER
CALL   BDOS
MOV    A,H      ;TEST IT FOR NON-ZERO (2.0)
ORA    L
RNZ             ;WE'RE DONE IF 2.0
LHLD   BDOS+1   ;FIND BDOS ENTRY POINT
LXI    D,9      ;OFFSET TO SECTRAN ROUTINE
```

```
DAD      D
XCHG               ;GET SECTRAN ADRS INTO DE
         LXI       H,SECTRAN ;POINT TO SECTRAN JUMP
                             ;IN OUR LOCAL JUMP TABLE
MVI      M,0C3H    ;STORE A JUMP INSTRUCTION
INX      H
MOV      M,E       ;PUT SECTRAN ADRS INTO JMP INST.
INX      H
MOV      M,D
RET
```

The local area set up to receive the table would look like this:

```
MYTABL   EQU    $      ;define local BIOS jump table
CBOOT:   DS     3      ;cold-boot entry point
WOOT:    DS     3      ;warm-boot entry
CONST:   DS     3      ;console-status test
CONIN:   DS     3      ;console input
CONOUT:  DS     3      ;console output
LIST:    DS     3      ;list output
PUNCH:   DS     3      ;punch output
READER:  DS     3      ;reader input
HOME:    DS     3      ;home disk drive head
SELDSK:  DS     3      ;select disk drive
SETTRK:  DS     3      ;select track
SETSEC:  DS     3      ;select sector
SETDMA:  DS     3      ;set disk transfer address
READ:    DS     3      ;read a sector
WRITE:   DS     3      ;write a sector
LISTST:  DS     3      ;list status test
SECTRAN:DS      3      ;perform sector translation
```

After the table has been moved, any BIOS routine can then be called by name.

LOCATING THE DPB

The Disk Parameter Block present under CP/M-80 1.4 is located shortly after the beginning of the BDOS, and can be found by adding an offset to the address field of the system entry point at location 5. The following code will set up a pointer in HL to the 1.4 DPB:

```
LHLD     6        ;address field of CP/M-80 entry
LXI      D,52     ;DPB occurs 52 bytes after entry
DAD      D        ;add the offset to entry
```

CP/M-80 2.0 allows access to the DPB via a system call. The following code can be used in place of that given previously, when operating under CP/M-80 2.0:

```
MVI      C,31     ;"get dpb" system function number
CALL     5        ;system entry point
```

Quite often, it is necessary to access the disk while avoiding BDOS function calls (e.g., when accessing the BIOS disk drivers directly). In this case, the DPB may be located by using the BIOS SELDSK call. SELDSK under CP/M-80 2.0 returns the address of a data structure called the Disk Parameter Header (DPH) in the HL register pair (or the value 0 if the drive sent to SELDSK doesn't exist). The DPH contains several items relating to the drive, two of which are useful for direct disk addressing: the sector translation table address (the first item within the DPH) and the address of the DPB (offset 10 bytes into the DPH). The following code illustrates direct BIOS access to these parameters:

```
MVI      C,1      ;requesting drive "B" in this example
CALL     SELDSK;call the BIOS routine directly
MOV      A,H      ;test for HL=0 (select error)
ORA      L
JZ       SELERR ;select error branch to error handler
MOV      E,M      ;fetch addrs of sector translate table
INX      H
MOV      D,M
XCHG              ;save sectran table address
SHLD     SECTBL
XCHG
LXI      D,9      ;remaining offset to DPB address DPH
DAD      D        ;add in the offset
MOV      A,M      ;get lo byte of DPB address
INX      H        ;point to hi
MOV      H,M      ;fetch hi byte
MOV      L,A      ;HL now has pointer to DPB
```

The best way of deciding *how* to access the DPB is to include both routines in your program, and link them together with a CP/M-80 version number system call. This is illustrated in the sample disk login routine of Listing 1.

USING THE DPB

Now that we know how to access the DPB (and other essential elements of the BIOS), it's time to learn how to use it. I'll do this by defining several common problems in direct-disk accessing programs, and show how these problems can be solved using the parameters supplied by the DPB.

The most fundamental task we'll examine is the conversion of block numbers to physical track and sector numbers. This is necessary for finding the physical end of the disk (which is almost always necessary when writing programs that directly access the disk), and is also very handy when doing selection by block number (a function provided by several disk utility programs, most notably Ward Christensen's DU.ASM.)

Track and sector can be derived from the following equations:

$$\text{Track} = ((\text{SPB}*\text{BLN})/\text{SPT})+\text{RESTRK})$$
$$\text{Sector} = ((\text{SPB}*\text{BLN})\ \text{MOD}\ \text{SPT})+1$$

where

BLN	= Block number
SPB	= Sectors per block
SPT	= Sectors per track
RESTRK	= Sectors per track

The SPT value is explicitly defined in the Disk Parameter Block, so the only question here in writing an actual conversion routine is determining the SPB value. It so happens that the BLKSHF and BLKMSK values determine implicitly the

(continued next page)

blocksize, as shown in Table 1. Referring to the table, note that the SPB is always an integral multiple of 8 (which just happens to be 2^3). As you can see, the BLKSHF value is the base two logarithm of the number of sectors per block, which simplifies the multiplication in the above equations to a series of simple (and fast!) left shifts.

Now we're ready to build the block-to-track-and-sector conversion routine. The argument to this function (the block number) will be passed in the BC register pair, and it will return with the track number in HL and the sector number in DE. We will use the parameters SPT and BLKSHF from our local copy of the DPB (which we moved to a local area in the previous routine). Here then is the code:

```
BLKCNV: MOV     H,B     ;MOVE ARG TO HL
        MOV     L,C     ;FOR EASY SHIFTING
        LDA     BLKSHF  ;FETCH LOG2 (SPB)
SHIFT:  DAD     H       ;THIS LOOP EFFECTIVELY DOES
        DCR     A       ;  (SPB*BLN)
        JNZ     SHIFT
        XCHG            ;FREE UP HL,SAVING (SPB*BLN)
        LHLD    SPT     ;GET SPT FOR ((SPB*BLN)/SPT)
        MOV     A,L     ;DO TWO'S COMPLEMENT SO WE
        CMA             ;  DO DAD, WHICH IS FASTER
        MOV     L,A     ;  THAN SUBTRACTION
        MOV     A,H
        CMA
        MOV     H,A
        INX     H
        XCHG            ;DIVISOR IN DE,DIVIDEND IN HL
```

The next block of code implements division by repetitive subtraction. To save time, we are using the two's complement of the divisor, since a double add instruction (DAD) is much faster on an 8080 than a double subtract. The rest of the routine follows:

```
        LXI     B,0     ;INITIALIZE QUOTIENT
DIVLP:  INX     B       ;BUMP QUOTIENT
        DAD     D       ;SUBTRACT BY 2'S COMPL. ADD
        JC      DIVLP   ;TILL WE RUN OUT OF DIVISOR
        DCX     B       ;CORRECT OVERSHOOT IN QUOTIENT
        XCHG            ;CORRECT OVERSHOOT IN REMNDER
        LHLD    SPT
        DAD     D
        INX     H       ;MAKE SCTR # RELATIVE TO 1,NOT 0
        XCHG            ;SECTOR NUMBER IS NOW IN DE
        LHLD    RESTRK  ;COMPLETE TO TRACK EQUATION
        DAD     B       ;BY ADDING RESTRK TO QUOTIENT
        RET             ;TO GET ABSOLUTE TRCK # IN HL
```

Now let's build on that subroutine. We almost always have a need to determine the disk limits, so we can take advantage of the DSM (maximum block number) field of the DPB to determine the maximum track and sector values. Here is the code to accomplish that:

```
LHLD    DSM     ;FETCH MAXIMUM BLOCK NUMBER
MOV     B,H     ;MOVE IT TO BC
MOV     C,L
CALL    BLKCNV  ;GET IT'S TRACK AND SECTOR #
SHLD    MAXTRK  ;FIRST TRACK,
XCHG            ;THEN SECTOR
SHLD    MAXSEC
```

(Note that this code was included in our disk login routine).

Let me digress for a moment and mention that the sector number determined by BLKCNV is the *logical* sector number and may not (usually *will* not) correspond to the physical sector number on the disk. Before you try to use the BIOS, call SETSEC, you should first call SECTRAN to do the logical-to-physical translation. This is straightforward under CP/M-80 2.0, but 1.4's SECTRAN routine has the side effect of calling SETSEC itself, so any subsequent call to SETSEC should be inhibited.

Sometimes it is necessary to determine the block number, given track and sector. This is a bit more complex, since a given track and sector can fall anywhere within the associated block. To account for this, our sample code will determine a value called BLKDIS (block displacement). The combination of BLOCK and BLKDIS will then be unique for each track/sector combination. Both BLOCK and BLKDIS are defined in terms of track and sector as follows:

BLOCK = (sector+((track-restrk)*spt)/log2(SPB)
BLKDIS = (sector+((track-restrk)*spt)) AND (SPB-1)

We can now code the conversion routine; given the track number stored at CURTRK and the sector number at CURSEC, the following routine will return the block number in HL and the block displacement in A:

```
CALBLK: LHLD    RESTRK;DO (TRACK-RESTRK)
        XCHG
        LHLD    CURTRK
        CALL    SUBDE ;USING SUBTRCTION UTILITY
        XCHG          ;PUT (TRACK-RESTRK) IN DE
        LHLD    SPT   ;NOW MULT(TRACK-RESTRK) BY SPT
        CALL    MULT  ;USING MULTIPLICATION UTIL
        XCHG          ;PUT (TRACK-RESTRK)*SPB  IN DE
        LHLD    CURSEC;GET SECTOR #
        DCX     H     ;MAKE RELATIVE TO 0, NOT 1
        DAD     D     ;HAVE SEC+((TRK-RESTRK)*SPT))
        LDA     BLKMSK;GET SPB-1
        MOV     B,A   ;STUFF IT IN B
        MOV     A,L   ;LO BYTE SEC+((TRK-RESTRK)*SPT))
        ANA     B     ;DO 'AND' OP TO FORM BLKDIS
        PUSH    PSW   ;SAVE IT FOR NOW
        LDA     BLKSHF;GET LOG2(SPB)
        MOV     B,A   ;FORM A COUNTER IN B
CALP:   CALL    ROTRHL;DIV. BY 2 USING SHIFT RT UTIL
        DCR     B     ;ONCE PER BLKSHF
        JNZ     CALP  ;TO GET BLOCK NUMBER IN HL
        POP     PSW   ;RETRIEVE BLKDIS
        RET
```

(Note that the utility routines MULT, SUBDE and ROTRHL are given in Listing 2).

It is sometimes necessary to determine the size of the directory area of the disk (for memory allocation purposes, etc). This can be done using the DIRAL0 and DIRAL1 (directory block allocation) bytes from the DPB, but it would be better to use the DIRMAX value, since it defines the maximum usable directory space (which may be less than the space defined by the allocated blocks defined by AL0 and AL1). Recall that DIRMAX represents the highest numbered directory entry (with the first entry numbered zero). Since there are four 32-byte directory entries per disk sector, we can determine the directory size in sectors as follows:

```
LHLD    DIRMAX;GET MAXIMUM ENTRY NUMBER
INX     H      ;CONVERT IT TO NUMBER OF ENTRIES
CALL    ROTRHL;DIVIDE BY 4 BY SHIFTING RIGHT TWICE
CALL    ROTRHL
```

The routine ROTRHL (given in listing 2) simply shifts HL right once, effectively performing a divide by two.

We can write a more general-purpose directory sizing routine, that will return the size in both bytes (let's use HL for that) and sectors (in DE) as follows:

```
DIRSIZ: LHLD    DIRMAX;GET MAXIMUM ENTRY NUMBER
        INX     H      ;CONVERT IT TO # OF ENTRIES
        PUSH    H      ;SAVE THIS FOR SECTOR CALC.
        MVI     A,5    ;SHIFT COUNT FOR MULTIPLY x 32
DIRLP1: DAD     H      ;MULTIPLY # OF ENTRIES x  32
        DCR     A      ;  (ARE 32 BYTES/ENTRY)
        JNZ     DIRLP1;  TO GET SIZE IN BYTES
        XCHG           ;RETURN W/BYTE COUNT IN DE
        POP     H      ;GET # ENTRIES BACK
        CALL    ROTRHL;DIVIDE NUMBER OF ENTRIES BY 4
        CALL    ROTRHL;  (ARE 4 ENTRIES/SECTOR)
        RET            ;TO GET SECTOR COUNT IN HL
```

Our final routine is called NXTSEC, and can be used to increment the current track and sector values. This is useful in sequential processing, such as disk copying, or searching the disk for a given string. It uses the current disk address stored in TRACK and SECTOR, and updates them to the next sequential disk address. Also used is the maximum track number (MAXTRK) set up at disk login. It returns the carry flag set if the end of the disk has been reached, reset otherwise.

```
NXTSEC: LHLD    SECTOR;SET CURRENT SECTOR
                INX     H      ;BUMP IT ONE
        XCHG           ;PREPARE FOR (SPT-CURSEC)
        LHLD    SPT    ;FETCH SECTORS PER TRACK
        CALL    SUBDE  ;DO (SPT-CURSEC)
        XCHG           ;GET NXT SECTOR BACK IN HL
        JNC     NEXTOK;JUMP IF NO WRAP TO NEXT TRACK
        LHLD    TRACK  ;ADVANCE TO NEXT TRACK
        INX     H      ;BUMP THE TRACK NUMBER
        XCHG           ;PREPARE FOR (MAXTRK-TRACK)
        LHLD    MAXTRK;FETCH MAX TRK # (SETUP AT LOGI
        CALL    SUBDE  ;DO (MAXTRK-TRACK)
        RC             ;END OF DISK, RETURN CARRY SET
        XCHG           ;NO, GET NEW TRACK NUMBER TO HL
        SHLD    TRACK
        LXI     H,0    ;SET 0-RELATIVE SECTOR NUMBER
NEXTOK: SHLD    SECTOR;SET SECTOR NUMBER
        RET            ;(CARRY CLEAR IF WE GOT HERE)
```

DISCLAIMER

I must point out that certain double-density floppy disk systems have one or more of the system tracks formatted in single-density, in order to accommodate boot loaders in ROM. Further, some controllers may require single-density formatted disks to have system tracks formatted double-density, to allow more space for large versions of the CBIOS. Since CP/M-80 normally never accesses these tracks, this is normally not a problem. However, programs that access the disk directly through the CBIOS may not perform correctly when trying to read or write these tracks.

References

1) These utilities are available from most RCPM systems around the country:

a. DU.COM - Ward Christensen's Disk Utility. Certainly one of the finest programs of its kind available for any price, and it's FREE! DU allows you to read and write any sector of the disk, display disk sectors in hex and ASCII, search the disk for a string, modify sectors in hex or ASCII, map the directory by block number, position to block, track and sector, view the disk as a text file, and many, many others.

b. FINDBAD.COM - by Gene Cotton, originally published in Interface Age, September, 1980. This is a disk test utility that "locks out" bad (unreadable) sectors found on the disk, and reports its progress on the system console.

c. SAP.COM - by L. E. Hughes. Sorts and packs the disk directory. Deletes zero-length files, and re-initializes the unused directory space to hex E5's.

2) Certain names of disk parameters are different from what is shown in the CP/M-80 system alteration guide. I've done this for clarity; specifically, DRM was renamed to DIRMAX, BSH to BLKSHF, BLM to BLKMSK, EXM to EXTMSK, AL0 to DIRAL0, AL1 to DIRAL1, CKS to CHKSIZ, and OFF to RESTKS.

Table 1

CP/M 1.4		CP/M 2.0	
SPT	(byte)	SPT	(word)
DIRMAX	(byte)	BLKSHF	(byte)
BLKSHF	(byte)	BLKMSK	(byte)
BLKMSK	(byte)	EXTMSK	(byte)
DSM	(byte)	DSM	(word)
DIRALO	(byte)	DIRMAX	(word)
RESTRK	(byte)	DIRAL0	(byte)
		DIRAL1	(byte)
		CKSIZE	(word)
		RESTRK	(word)

Table 2

BLOCKSIZE IN BYTES	SECTORS PER BLOCK	BLKSHF	BLKMSK
1024 (1k)	8	3	7
2048 (2k)	16	4	15
4096 (4k)	32	5	31
8192 (8k)	64	6	63
16384 (16k)	128	7	127

(continued next page)

Listing 1

```
;
; sample disk-login routine...this routine selects the disk in the C
; register, determines the operating system version number, and moves
; the disk parameters to a local storage area
;
LOGIN:  CALL    SELDSK  ;login the drive
        PUSH    H       ;save result (in case CP/M 2.0)
        MVI     C,12    ;system "return version" call
        CALL    BDOS    ;system entry (at CPBASE+5)
        MOV     A,H     ;test HL=0 (indicates version < 1.4)
        ORA     L
        POP     H       ;retrieve login result
        JZ      V14     ;go get DPB the 1.4 way
;
; we're running under CP/M 2.x
;
V20:    MOV     A,H     ;test for HL=0 (select error)
        ORA     L
        JZ      SELERR  ;select error; branch to an error handler
        MOV     E,M     ;get sector translation table address
        INX     H       ; into DE
        MOV     D,M
        XCHG            ;now save sec tran table address
        SHLD    SECTBL
        XCHG
        LXI     D,9     ;remaining offset to DPB address of DPH
        DAD     D       ;add in the offset
        MOV     A,M     ;get lo byte of DPB address
        INX     H       ;point to hi
        MOV     H,M     ;fetch hi byte
        MOV     L,A     ;HL now has pointer to DPB
        LXI     D,MYDPB ;now move all disk parms to local area
        MVI     B,15    ;make a counter out of DPB length
MOVDPB: MOV     A,M     ;now move the DPB byte-by-byte
        STAX    D       ;into local storage
        INX     H
        INX     D
        DCR     B
        JNZ     MOVDPB
;
; include the following 7 lines of code only if you need to
; establish the maximum track value (maxtrk).
;
SETMAX: LHLD    DSM     ;FETCH MAXIMUM BLOCK NUMBER
        MOV     B,H     ;MOVE IT TO BC
        MOV     C,L
        CALL    BLKCNV  ;GET IT'S TRACK AND SECTOR NUMBER
        SHLD    MAXTRK  ;FIRST TRACK,
        XCHG            ;THEN SECTOR
        SHLD    MAXSEC
;
        RET
:
; we're running under CP/M 1.4
;
V14:    LHLD    6       ;address field of CP/M entry
        LXI     D,52    ;DPB occurs 52 bytes after entry
        DAD     D       ;add the offset to entry
;
; moving the DPB is trickier under CP/M 1.4...many of
; the values are BYTE rather than WORD values, and we
; have to adjust
;
        MVI     D,0     ;get hi byte 0 to chg bytes to words
        MOV     E,M     ;get byte SPT value
        INX     H
        XCHG
        SHLD    SPT     ;store it as a word
        XCHG
        MOV     E,M     ;get byte DIRMAX value
        INX     H
        XCHG
        SHLD    DIRMAX  ;store it as word
        XCHG
        MOV     A,M
        INX     H
        STA     BLKSHF  ;BLKSHF and BLKMSK are bytes
        MOV     A,M
        INX     H
        STA     BLKMSK
        MOV     E,M     ;get byte DSM value
        INX     H
        XCHG
        SHLD    DSM     ;store as a word
        XCHG
        MOV     E,M     ;get DIRALO value
        INX     H
        XCHG
        SHLD    DIRALO
        XCHG
        MOV     E,M     ;and reserved tracks value
        XCHG
        SHLD    RESTRK
        JMP     SETMAX  ;go set maximum track value
;
; the following local storage for the disk parameter
; block should be included in your data area.
;
MYDPB   EQU     $       ;define location of DPB
;
SPT:    DS      2       ;sectors per track
BLKSHF: DS      1       ;block shift value
BLKMSK: DS      1       ;block mask value
EXTMSK: DS      1       ;extent mask value
DSM:    DS      2       ;max block number
DIRMAX: DS      2       ;max dir entry number
DIRAL0: DS      1       ;directory allocation 0
DIRAL1: DS      1       ;directory allocation 1
CHKSIZ: DS      2       ;checked directory entries
RESTRK: DS      2       ;number of reserved tracks
;
; storage for various disk address items
;
TRACK:  DS      2       ;current track number
SECTOR: DS      2       ;current sector number
MAXTRK: DS      2       ;maximum track number
MAXSEC: DS      2       ;maximum sector number
SECTBL: DS      2       ;adrs of CPM2 sector xlate table
;
;----------- END ------------
```

Listing 2

```
;
; Utility routines used with direct disk accessing
; programs to do disk arithmetic
;
;
; cheap (and not extremely efficient) multipication
; routine; does HL=HL*DE, using repeated addition.
;
MULT:   PUSH    B       ;DON'T ALTER BC AND DE
        PUSH    D
        XCHG            ;SWAP MULTIPLIERS
        MOV     B,D
        MOV     C,E     ;GET 1ST MULTIPLIER TO BC
        MOV     A,B     ;TEST FOR MULT BY ZERO
        ORA     C
        JNZ     MULCON  ;JUMP IF NOT MULT BY ZERO
        LXI     H,0     ;YES, GET QUICK ANSWER
        JMP     MLDONE
;
MULCON: DCX     B       ;ADJUST LOOP COUNT
        MOV     D,H     ;MOVE 2ND MULTIPLIER TO DE
        MOV     E,L
;
MULTLP: MOV     A,B     ;OUT OF MULT #1?
        ORA     C
        JZ      MLDONE  ;THEN GO RESTORE AND EXIT
        DAD     D       ;NOPE, DO ANOTHER ADDITION
        DCX     B       ;ADJUST MULTIPLIER #1
        JMP     MULTLP
;
MLDONE: POP     D       ;RESTORE REGS
        POP     B
        RET
;
; subtract de from hl with result in hl
;
SUBDE:  MOV     A,L     ;DO LOWER 8 BITS FIRST
        SUB     E       ;(IGNORE CARRY)
        MOV     L,A
        MOV     A,H     ;THEN UPPER
        SBB     D       ;(CARRY COUNTS NOW)
        MOV     H,A
        RET
;
; NEGATE HL (2'S COMPLEMENT)
;
NEG:    MOV     A,L     ;DO IT 8 BITS AT A TIME
        CMA
        MOV     L,A
        MOV     A,H
        CMA
        MOV     H,A
        INX     H
        RET
;
; DIVIDE HL BY TWO (SHIFT HL RIGHT ONE)
;
ROTRHL: ORA     A       ;CLEAR CARRY
        MOV     A,H     ;SHIFT UPPER 8 BITS
        RAR
        MOV     H,A
        MOV     A,L     ;THEN LOWER
        RAR
        MOV     L,A
        RET
```

Full Screen Program Editors, Part 1

Ward Christensen

Introduction

This article is the first in a series reviewing several popular "full screen" editors. I have specifically selected those which are more program-oriented than text-oriented.

Each of us has unique editing needs. Few people can truly say they do only "program editing". Program documentation falls more into the text category. I will thus comment on the text editing capabilities of the editors, while not concentrating on this factor.

I have had WordMaster for years, and more recently obtained MINCE, VEDIT, and PMATE. These are the editors I will review. If you have a favorite editor that's not among those, or have seen one you think should be reviewed, let me know about it via *Lifelines*.

Objectives

I will compare and contrast the editors, and give you a bit more technical insight into their strengths and weaknesses than you would get from reading an advertisement. In Part I of the review, I will establish criteria for evaluation. Subsequent parts will each cover a single editor, in terms of the established criteria.

Then, having the detailed editor reviews for background, a final part will concentrate on comparisons — how one editor accomplishes something that another does a different way. This will help you form your own opinions.

Background

My first microcomputer editor was the Processor Technology Software Package Number One, a 4K, 8080, resident editor and assembler. It allowed inserting and deleting lines by line number. Period. It got the job done. It served me well for about a year.

In late 1976, Robert Swartz convinced me to buy CP/M-80, and "sat me down" at his CRT to learn ED. I was very impressed, and continued to use ED for about 18 months.

In mid-1978, Chuck Douds suggested I get WordMaster, a full screen editor running under CP/M-80. I thought ED was good enough, but he eventually convinced me to buy it. Was I glad! Its command mode was almost identical to ED, and video mode was very easy to learn. It was fast, and bug free.

By mid-1981, although I was still very happy with WordMaster, other editors began to catch my eye. MINCE, with its promise of split-screen multi-file editing, sounded like what I needed in preparing CPMUG abstracts. Abstracts are developed from my personal thoughts, the author's own .DOC files, a CPMUG contribution form, and possibly from comments at the front of the source programs themselves.

I began to think about doing a "full fledged review" of WordMaster, MINCE, and other editors. I purchased MINCE, and was completely intrigued with its capabilities. I set out to purchase VEDIT, but the place I talked to insisted I must buy two separate full-price licenses, in order to obtain both a memory mapped, and a CRT based version.

I subsequently called CompuView, and found a second license was available for only a nominal update fee. They then graciously gave me copies for evaluation, and in return, I purchased the $95 upgrade to obtain the 8086 version.

I contacted *Lifelines* about doing a side-by-side editor review, and they offered to provide an evaluation copy of PMATE. There have been excellent reviews of some of the editors: Barry Dobyn's review of MINCE in Dr. Dobbs' Journal, April 1981; Harris Landgarten's review of PMATE in the April 1981 issue of *Lifelines;* and Christopher Kern's review of MINCE in the September issue of BYTE. I urge you to consult them for further insight.

Evaluation Criteria

If you were to judge an editor, there would be only one important criterion: do you like it or not. That however does not make for an interesting and informative review. It also does not account for "taste" — my likes and dislikes are certainly not identical to anyone else's.

Here are the criteria I feel are important in selecting an editor. They are the ones I will refer back to in the individual editor evaluations.

Subjective Criteria

DOCUMENTATION: The documentation should serve four purposes: (1) an installation guide; (2) an introduction for the beginner; (3) a learning tool for the advanced user; and (4) a reference manual.

SPEED: Although this is not the ultimate criterion, the editor should not "slow you down". It should take advantage of "modern" terminal characteristics, such as line insert and line delete. This significantly improves usability.

ERGONOMICS: It should be "comfortable" to use. There should be no "frustration" introduced by using it. Frequently used functions should not make you feel you are performing "finger exercises".

CONFIGURABILITY: It should be able to be modified (1) for a variety of terminal types; (2) for different keyboard layout preferences. The latter is not mandatory, but it is nice to be able to configure the editor to your preferences.

(continued next page)

EASY TO LEARN: If you have to keep referring back to some manual, you just won't be productive. When you do refer to the manual, it should be easy to go right to what you were looking for.

Objective Criteria

Video related criteria

FULL SCREEN: Move the cursor about the screen, both in entry mode, and for making corrections.

SCROLLING: Move up or down within the document, line by line, or screen by screen. If the file doesn't all fit in memory, it should still "appear to", with automatic disk buffering.

INSERT: "Squeeze" in characters or lines in, without explicitly having to "open up" space for them first.

OVERTYPE: Move to a mistake, and overtype the correction.

UNDO-KEY. (Sometimes called the "OH #$%&" key). If you have accidentally deleted a line, or block, it is nice to be able to undo the delete. This serves a double purpose of being an easy- to-use "move" command, since you can delete something, position the cursor elsewhere, and bring it back.

REPEAT KEY: The key you press following the repeat key will have its action repeated several times. The repeat key lets the editor do some of the work for you when you want to do something simple over and over, such as move down several lines, or insert multiple lines.

TEXT EDITING ABILITIES: Not critical to program editing, but mentioning these capabilities will help you judge which editor best suits your mix of editing requirements.

The simplest editor can probably perform word tab and back tab, and perhaps word delete.

Command related criteria

MOVE: Readily moving the pointer to where you want to be in the file, by going ahead or back some number of characters or lines.

DELETE: Delete characters or line, ahead or back. A block marking and deleting mode is most helpful.

INSERT: Add characters, including CR/LF combinations.

TYPE: Allow you to "follow the progress", or see the results of your changes. For a fast memory mapped terminal, the update may even be shown full-screen.

FIND: Find character strings, such as to locate a label, etc.

CHANGE: Make non-video changes, such as to change all occurrences of a particular label, to something else. This is better done in a "command mode", rather than having to search for, and change, all the strings in video mode.

MOVE and COPY: Allow moving or copying some arbitrary part of a file to another place in the file. Useful, for example, when two subroutines are quite similar, and it would save lots of typing to be able to make a copy of one, and just change the parts that need it.

Moving and copying should ideally be done by "marking" the text in some way.

COMMAND STRINGS: Putting together all the above abilities into "little programs". Preferably, each of the above commands should be able to be modified by a number, such as to find the third occurrence of something, or to move down 27 lines. Furthermore, the command strings should have an overall repeat ability, either limited, or unlimited. Nesting, in which you say "I want to do such-and-such 4 times, and within that, something else, 7 times", should be supported.

MULTIPLE EDITS: If you have a lot of edits to do, it is nice not having to exit the editor, only to re-execute it for some other file.

File related criteria

BACKUP: The editor should create backup copies of what you are editing, so that if you later realize you have made a major mistake, you can erase the new version and rename the .BAK file.

The backup file ability should have some means to compensate for its size, such as allowing it to be placed on another disk.

SAVE: Write your changes back to disk, and continue editing.

QUIT: Give up if you have made a gross mistake. The editor should probably prompt you to ensure you *really* meant to quit.

READ: Bring in all, or preferably, a piece, of another file.

WRITE: Write out some piece of the file being edited, to another file.

DIRECTORY: Access to the CP/M-80 directory is helpful. For example, you may want to read in part of another file, but can't remember its name.

It would also be nice to erase files (to make room), and perhaps to even log in a new disk, or change disks to get at some piece of data on a disk that is not on line.

Gift Subscriptions

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

Your name and address:

Name ______________________
Address ______________________
City ____________ State ______ Zip ______

Shirt size __________
□ Check enclosed
VISA or MasterCard Number
Expiration Date

____

____
Signature (if payment is by credit card)

The name and address of the gifted one:

Name ______________________
Address ______________________
City ____________ State ______ Zip ______

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

KIBITS

MIKE
THAT'S THE LAST TIME I GO TO BED WITHOUT BACKING UP MY FLOPPY!
Seton & Bess 1982

T/MAKER II: A Continuing Review

Raymond J. Sonoff

The first article in this series presented a background on T/MAKER II, some of the initialization steps (and problems) that surfaced when I began to install T/MAKER II on my CP/M-based system, and highlighted some of the functions, utility files and possibilities for applying this product.

This particular article presents some of my initial observations and conclusions after having gone through the Tutorial section of the T/MAKER II manual followed by creation of several files that utilized both text editing and data calculating features and functions of T/MAKER II.

Examining T/MAKER II as a tool for accomplishing tasks that would normally require sophisticated word processing and/or extensive experience in writing programs in BASIC has convinced me of its having powerful features that should be pointed out to anyone who is going through any decision-making process, or who is in a quandary as to what software package to consider for a system. Perhaps you will be convinced to try T/MAKER II after noting several particularly significant findings I have discerned from my usage of T/MAKER II to date:

1. T/MAKER II (version of 2.3.2. of T/MAKER) incorporates a text editor that proves to be essentially compatible with those found in either MAGIC WAND or WORDSTAR. (Carriage returns embedded in the text do have to be removed when justified text or a different page width is called for, namely one greater than that used in creating the original working file. Otherwise, the associated print formatting commands will not be able to function as desired for either software package. Fortunately, this task is a rather simple one to implement on any T/MAKER based file.) In short, Working Files created with T/MAKER II can be processed by two outstanding word processing "packages" when you want to incorporate the files of T/MAKER into either word processor. Moreover, this process works in the other direction as well. If you, for example, have computations to make on a file you created some time ago and for which the data, rates, etc. have changed, you can edit the file — now embedding the appropriate T/MAKER II Symbols, Example Line, Zero Values Line, Row Equations, Column Equation Control Codes, etc. — and be able to generate an updated file complete with computations. Moreover, you can create MASKS which are files that are used by the LOAD and UNLOAD functions of T/MAKER II to either load a complete DATA File into a Working File, or to Unload a Data File from a Working File. You can even COMBINE Tables of Data into a new File. You no doubt can see how extremely powerful a tool T/MAKER II can prove to be. That an individual need not be a sophisticated programmer to quickly put this product to use is the very best feature of all.

2. T/MAKER II offers, in general, the easiest-to-use editor from among several that I have used. This includes the three CP/M-based editors: MAGIC WAND, WORDSTAR, and the ED.COM utility file; an early day utility called TED that was associated with K2FDOS; and also an editor that was incorporated in a Signetics' TWIN microprocessor development system that was nearly identical to that found in Tektronix 8002 development systems. In short, T/MAKER II has downright spoiled me, at least whenever the application or task entails just slightly more than straightforward word processing! Example of these situations include numerical/character movement and/or sorting, or combinations involving calculations and movements of columns, etc. Of course, there are some minor inconveniences to be noted, but they are far outweighed by the numerous powerful capabilities that are generally available simply by entering particular keystrokes. Oh yes, even the keystrokes can be user-defined.

3. Table of readily formatted calculations along with associated headings and any desired supporting text are far and away much easier to create and to modify than would be possible using the conventional BASIC programming approach. Consider the following facts: 1) You can actually SEE (and even store away as separate files on diskette) any initial, intermediate, and/or final Table layouts that you do create. By comparison, if you did this same task using BASIC language, you would have to create (and, therefore, thoroughly understand) a detailed program using BASIC to accomplish the same end result. Of course, knowing how to use BASIC required you to document, debug, and mentally convert your line after line of BASIC REM statements, DATA READ (statements, PRINT x number of spaces, etc., DO LOOP arrangements (including the assigning of index variables, etc. etc.) until you thought you had something that would work. Then you would go through a trial-and-error testing process. And so it would go.... HOW BORING anyway!! 2) Not only can you define the layout but also the types and sequences of calculations to be made can be readily changed using the editing functions of T/MAKER II. 3) You can also use the MASK function to either LOAD or UNLOAD data associated with a created file. 4) Six conditional COMPUTE categories can be specified for controlling column equations. These range from "always compute" through "just compute" and even to "suspend compute". 5) Reasonable short setup times for creating tables—even when the table is of moderate complexity should be achieved by a T/MAKER II user. For once he has performed even a few relatively simple examples or created some relevant sets of tables, be they simple checkbook balancing, an alphabetical or numeric sorting, combining of two or more sets of data, cash flow analyses, etc. the user will become quite at home with the methodology of T/MAKER II. He will find that T/MAKER II will allow him to quickly yet effectively solve, document, and report and/or retrieve results as needed.

4. Yes, some negatives to T/MAKER II's Editor that might appear as oversights are the following: absence of both "delete word to the right of the cursor" and "move cursor to the left one word" commands, inability to provide a "characters-per-inch" command that T/MAKER II could use as in conjunction with a printer (such as a Diablo, Qume, or NEC unit) having proportional printing capability, and no PUT command that allows you to output a portion of the Working File to a diskette, and to an awkwardness in trying to use a JOIN/BREAK LINES procedure to achieve phrases that remain within one FRAME (another mode that you will enjoy having as a working tool when large tables of data have to be manipulated!).

That these specific commands may or will be incorporated in subsequent revisions of T/MAKER II need not be of immediate concern, however. For, would you believe, some user generated solutions could be implemented as intermediate cures for at least some of these negative findings?

For example, by using easily created MACROS, a "FIND AND REPLACE string" could be made by combining the already existing "FIND string" and "REPLACE string 1 string 2" commands. Such a macro could be added as a "filename.MAC" file to the T/MAKER II 'working diskette' file library, and it could be INSERTed into the Working File as a "GET filename.MAC" instruction.

The highly desirable proportional printing feature could be achieved if you happen to employ either WORDSTAR or MAGIC WAND in your normal word processing activities. For example, if you so desire, you could edit the T/MAKER-generated file using one of these two word processing systems and embed the appropriate printer characters-per-inch command(s) in the now "hybridized" T/MAKER-generated file. Or, you might consider issuing the desired printer-related command(s) immediately prior to actual print operation initiation. The latter approach keeps the T/MAKER-based file 'clean' and readily usable by numerous word processor "packages".

5. Still more features worthy of mentioning are AUTOPAGE, INSERT, and FRAME MODES, but these will have to wait for another time for description. REDRAW SCREEN, GLOBAL REPLACE, and display WORKING FILE INFORMATION are other capabilities that must wait for another day....

LOOKING AHEAD

Practical examples and illustrations of some of the powerful operations available in T/MAKER II will begin with a forthcoming article in this series.

C-bits (All About BDS C version 1.45)

Reported by Bill Norris

And here's the bit-map: 1) bug fixes, 2) new features, 3) new linker, 4) sieve.doc.

Bug Fixes appear under New Versions in this issue.

New Features

CLINK will now recognize DEFF3.CRL as an automatic library file (not included). Use it for your custom functions, as DEFF.CRL and DEFF2.CRL are getting to be rather full. CLINK will now search all 3 DEFF files (if they exist) if a carriage return is typed in interactive mode. Previously, only DEFF.CRL was searched.

The special case handler for the code generator has been improved to more efficiently handle binary relational operations where exactly one of the operands is a constant. The operators affected are:"<", ">", "<=", ">=" "==", and "!=".

Two new functions have been added to the standard library:

```
int setjmp(buffer)
char buffer[JBUFSIZE];

longjmp(buffer,val)
char buffer[JBUFSIZE];
```

When "setjmp" is called, the current processor state is saved in the JBUFSIZE-byte buffer area whose address is passed as the argument ("JBUFSIZE" is defined in BDSCIO.H), and a value of zero is returned. Whenever a subsequent "longjmp" call is performed (from ANYWHERE in the current function or any lower-level function) with the same buffer argument, the CPU state is restored to that which it was during the "setjmp" call, and the program behaves as if control were just returning from the "setjmp" function, except that the return value this time is "val" as passed to "longjmp". A typical use of setjmp/longjmp is to exit up through several levels of function nesting without having to return through EACH level in sequence, to make sure that a particular exit routine (e.g., the directed I/O "dioflush" function) is always performed.

A New Linker

A new linker for BDS C called "L2" (a substitute for CLINK.COM) is now available from the BDS C User's group. L2, written by Scott Layson (of Mark of the Unicorn) in BDS C, has several interesting features:

L2 can link programs that are up to about 8K larger than CLINK: if there isn't enough room in memory to hold the entire program while building an image in memory, L2 performs a disk-buffering second pass. This means that the

(continued next page)

resulting COM files can be as large as the entire TPA on the target machine.

The number of functions per program is no longer limited to 255. While CLINK uses jump tables at the beginning of functions, L2 totally eliminates the jump tables and instead generates direct external call. This shortens programs by anywhere from 3% to 10%, and also speeds them up a little. Since the L2 source code (in C) is included, you can customize it yourself.

The L2 package also includes an overlay generator and documentation. It is available to BDSCUG members for the nominal cost of media and shipping (currently $8). BDSCUG members receive a newsletter approximately 6 times per year, and are entitled to compiler updates and library disks for low prices (typically 8$/disk). For information, contact: BDS C User's Group, Robert Ward, Coordinator, Dedicated Micro Systems, Inc. 409 E. Kansas Yates Center, Kansas 66783 (316) 625-3554.

SIEVE.C is included with the new version (1.45). This is the Sieve of Eratosthenes benchmark from BYTE, Sep. '81, pg. 186. This program is a bit cryptic, and as the output appeared incorrect, and a copy of the Sept. 81 BYTE was not immediately available, I called Leor Zolman, who promptly checked it out, found it to be correct, and deciphered it. A fragment from the program and explanations is shown in Figure 1 below.

The DEFINE statements indicate that the search for primes will be made for numbers up to TWICE the value specified in the DEFINE statements. The array elements of flags[] map onto the actual numbers as follows:

0=3, 1=5, 2=7, 3=9,...

So, to print out the results (ODD primes), the following can be added to the end of SIEVE.C:

```
for (i = 0; i<=SIZE; i++) {
        if (flags[i]) {
        printf("\n#%d = %d found to be prime.",i,i*2+3);
        }
}
```

```
#define SIZE 8190
#define SIZEPL 8191            LISTING OF SIEVE.C

                count = 0;
                for (i = 0; i <= SIZE; i++)
                        flags[i] = TRUE;
                for (i = 0; i <= SIZE; i++) {
                        if (flags[i]) {
                                prime = i + i + 3;
                                k = i + prime;
                                while (k <= SIZE) {
                                        flags[k] = FALSE;
                                        k += prime;
                                }
                                count++;
                        }
                }
```

That's all folks...

Change of Address

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

New Address:

NAME

COMPANY

STREET ADDRESS

CITY STATE

ZIP CODE

Old Address:

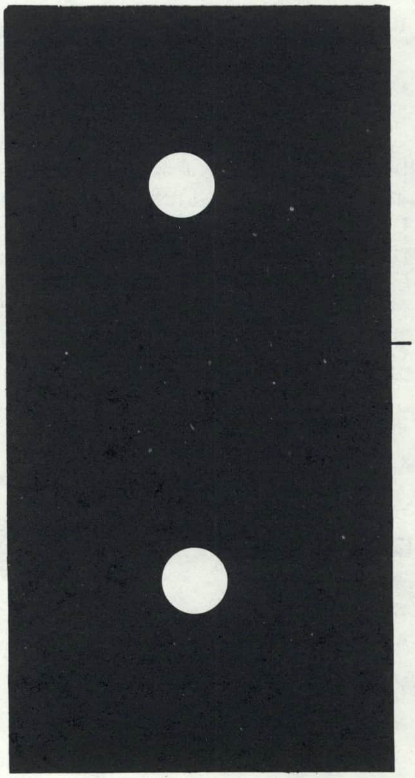
NAME

COMPANY

STREET ADDRESS

CITY STATE

ZIP CODE



New Versions

BDS C Compiler
Version 1.45

Here are the bug fixes and extensions available in this update.

1. Expressions of the form

   ```
   !(expr||expr)
   ```

   or

   ```
   !(expr && expr)
   ```

   may not have worked correctly when a VALUE was required for the expression; i.e., when used in some way other than in a flow control test. For example,

   ```
   x = !(a||b);
   ```

   might have failed, but

   ```
   if (!(a||b)) return 7;
   ```

   would have worked, since the expression was used for flow control.

2. Declarations of pointer-to-function variables for functions returning a CHARACTER value caused only one byte of storage to be reserved for the pointer, instead of two bytes (all pointers-to-functions require two bytes of storage, by virtue of being pointers). For example, in the sequence:

   ```
   char c1, (*ptrfn)(), c2;
   ...
   ptrfn = &getc;
   ```

   the assignment to 'ptrfn' would have incorrectly overwritten the 'c2' character variable, since only one byte would have been reserved on the stack for the 'ptrfn' variable while the assignment operation would have assumed there were two bytes reserved.

3. A bug in the ternary operator evaluator (?: expressions) caused the high-order byte of a 16-bit result to be incorrectly zeroed in the following situation: given a ternary expression of the form

   ```
   e1 ? e2 : e3
   ```

   where 'e2' evaluated to a 16-bit value (int, unsigned or pointer) and 'e3' evaluated to a character value (type char only), the entire expression was treated as having type char...so if 'e1' was true and 'e2' was bigger than 255, then the value of the expression ended up as only the low-order byte of the value of 'e2'. For version 1.45, whenever 'e2' and 'e3' do not BOTH evaluate to character values the type of the overall expression is guaranteed not to be char.

4. A sequence of two '!' (logical 'not') operators in a row did not always produce the correct result in an expression. For example,

   ```
   x = !!n;
   ```

   (convert n to a logical (0 or 1) value) might have produced the wrong result (0 instead of 1, or vice-versa).

5. A stack-handling bug in CC2 caused problems at run time when a sufficiently complex subexpression appeared in any but the final position of an expression involving the comma operator (","). For example, the following statement would not have worked correctly:

   ```
   for (i = 0; i < 10; x += y, i++)...
   ```

6. CC1 has not been recognizing illegal octal character constants as such; digits such as '8' and '9' within an octal constant will now draw an error in cases where they would have been ignored before. Also, certain other forms of illegal constants (aside from character constants) are now better diagnosed than before.

7. One more case has been found where an internal table overflow during code generation was not detected, causing the final command file to bomb as soon as it was executed (either by crashing the machine or immediately rebooting.) This occurred when a single large function containing many string constants was compiled. All fixed now.

8. The ↑Q-CR sequence required by CLINK in interactive mode (to abort the linkage in progress) cannot be typed in under MP/M systems, since ↑Q is used to detach a process. If you are running MP/M, then just type control-C instead of ↑Q-CR; this will also work for CP/M systems...the only difference is that when ↑Q-CR is used, then any currently active "submit file" processing is automatically aborted by CLINK before returning to command level, as a convenience. Under MP/M, it is necessary to type characters quickly at the keyboard (after ↑C-ing CLINK) to abort any pending submit file activity.

9. A slight bug in CLIB.COM (The C Library manager program) made it hard to exit CLIB from within a submit file (assuming XSUB is in use). The problem was that CLIB requires a confirmation character, 'y', to be typed after the 'quit' command is given. CLIB was getting the confirmation character by doing a single direct BDOS console input call, which required the user to manually type in the letter before any pending submit file processing could continue. This has been fixed by having CLIB get an entire line of input (using BDOS call 10) when seeking a confirmation; now the 'y' may be inserted into submit files. Note that the 'quit' command and the 'y' confirmation must be placed on separate consecutive lines in the submit file. If not using a submit file, the only difference is that now a carriage-return is required after typing the 'y'.

   Another minor problem with CLIB: function names longer than 8 characters were not being truncated when entered for operations such as renaming, resulting in too-long CRL file directory entries. All names are now properly limited to 8 characters.

10. A problem with file I/O under MP/M Version II has come up: The run-time package routine "vclose", called by the library function "close" whenever a file needs to be closed, has been optimizing for files open only for reading by *not* actually performing a "close" operation through the BDOS. This worked fine under CP/M, because CP/M didn't care whether or not a file that has had no changes made to it was ever closed; MP/M II, on the other hand, *does* seem to want such files to be explicitly closed; so by running many programs that didn't close their Read-only files, BDS C programs eventually caused MP/M to not allow any more files to be opened.

    This problem has been fixed by adding a conditional assembly

(continued next page)

symbol, called "MPM2", to the CCC.ASM source file. If you are running under MP/M II, you should set the "MPM2" equate to true (1) and reassemble CCC.ASM, yielding a new C.CCC after loading and renaming (you should only need ASM.COM for this, although MAC.COM works also). The change does *not* affect the size of C.CCC, so the libraries do not have to be reassembled as is usually the case when the run-time package is customized. The change simply causes a single conditional jump to be turned into three nop's, so that *all* files are always closed, instead of only the ones open for writing.

11. A bug was found in the '—scn' library function (affecting 'scanf'): when a lone carriage-return (newline) was typed in response to a "%s" format conversion, the format conversion was totally ignored. This caused the target string to remain unchanged from its previous contents, instead of correctly having a null string (consisting of a single zero byte) assigned to it.

12. A bug was found in the '—spr' library function (affecting 'printf', 'sprintf', and 'fprintf'): The default field width value was 1, causing a null string to be printed as a single space when the standard "%s" format conversion was used. For example, the statement:

    printf("Here is a null string: \"%s\"\n","");
    would have produced the output:
    Here is a null string: " "
    instead of:
    Here is a null string: ""
    The default field width value has been changed to 0, so null strings will now print correctly. An explicit field width may always be given in any format conversion, of course.

13. When the library function "sprintf" (formatted output directly into a memory buffer) is used, a null byte is appended onto the end of the output text.

14. In several library functions, as well as at one point in the run-time package, calls were made to BDOS function number 11 (interrogate console status) followed by an "ani 1" instruction to test bit 0 of the value returned by BDOS. On some systems, testing bit 0 is not sufficient since sometimes values other than 0 and 1 (or 0 and 255) are returned. *So*, all such sequences have been changed to do an "ora a" instead of an "ani 1", so that a return value of exactly 00h is interpreted as "no character ready" and any other value is interpreted as "yes, there is a character ready". The library functions that were modified this way are: 'kbhit', 'putchar', 'srand1', 'nrand', 'sleep' and 'pause'. The sequence to clear console status in the run-time package (CCC.ASM), near the label "init:", has likewise been changed (but a "nop" instruction was added to keep all addresses consistent with earlier versions of the run-time package.)

15. When customizing the run-time package (CCC.ASM) with the "cpm" symbol equated to zero, several symbols (named "SETNM" and "SETNM3", at the routine labeled "PATCHNM") were undefined; this has been fixed by adding some conditional assembly directives to insure that the labels in question are not referenced under non-"cpm" implementations, while the total code size remains constant so that the addresses of later run-time package utility subroutines stay exactly the same for all implementations.

16. A problem with the "bdos" library function has come up that is rather tricky, since it is system-dependent: A program that runs correctly under a normal Digital Research CP/M system might *not* run under MP/M or SDOS if the "bdos" function is used. A typical symptom of this problem is that upon character output, a character on the keyboard needs to be hit once in order to make each character of output appear.

    Normal CP/M behavior (which the C library function "bdos" had always assumed) is for registers A and L to contain the low-order byte of the return value, and for registers B and H to contain the high order byte of a return value (which is zero if the return value is only one byte). The CP/M interface guide explicitly states that "A = = L and B = = H upon return in all cases". Just in case CP/M 1.4 or some other system doesn't put the values in H and L from B and A, the "bdos" function copy register A is put into register L and copy register B into register H, to make sure the value is in HL (where the return value must always be placed by a C library function.)

    Not all systems actually follow this convention. Under MP/M, H and L always contain the correct value but B does not! So when B is copied into H, the wrong value results. So, the way to make "bdos" work under both CP/M 2.2 and MP/M was to discontinue copying B and A into H and L, and just assume the value will always be correctly left in HL by the system. This was done for v1.45.

    The way I left "bdos" for version 1.45 was so that it works with CP/M and MP/M (i.e., no register copying is done at all...HL is assumed to contain the correct value). This, of course, won't work in all cases under SDOS and perhaps other systems ...in those cases, you need to either use the "call" and "calla" functions to perform the BDOS call, or create your own assembly-coded version(s) of the "bdos" function (with MAC.COM, CMAC.LIB and BDS.LIB) to perform the correct register manipulation sequences for your system. Note that it may take more than one such function to cover all possible return value register configurations.

17. The "creat" library function had been creating new files and opening them for writing ONLY; this caused some confusion, so 'creat' has been modified to open files for both reading *and* writing following creation.

18. The "execv" function has been changed to return ERROR (-1) on error, instead of forcing an error message ("Broken pipe") to be printed to the standard error device.

19. The DIO (directed I/O and pipes)

package contained an obscure bug: if a pipe operation was aborted before completion, leaving a "TEMPIN.$$$" file in the directory, then the next pipe operation performed had gotten its own output mixed up with the output of the aborted pipe...the old output was used as input to the new next command, and the new output was lost. The new DIO.C has been fixed. (Note: DIO.C has also been slightly changed to properly interact with the new version of the "execv" library function.)

Another change has been made to the DIO package: the "getchar" function, when used without input redirection to read characters directly from the console, had not allowed for line editing in previous versions. Each character was obtained by a direct BDOS call and none of the special line editing characters (delete, ↑R, ↑U, etc.) were recognized. For version 1.45, an optional line buffer mechanism has been added to the DIO package so lines of console input can be fetched at one time by using the "read console buffer" BDOS call and all editing characters now function as expected. Operation of the package using buffered console input is still the same as before, except for one thing: to enter an end-of-file character (control-Z), it is now necessary to also type a carriage-return after the control-Z.

To enable console input buffering when using the DIO library, it is necessary to un-comment a line in the DIO.H file and re-compile DIO.C.

CP/M baZic II
Version 3.03

Version 03/03 of CP/M baZic has now been released. This version fixes several bugs or problems that were found in 02/02. In addition, a new feature has been added.

CP/M baZic now has a new function which is called CPMFN (CP/M Function). This function has been added to allow the baZic programmer to call any of the CP/M system function calls. The command syntax is:

T=CPMFN(<C ARGUMENT> [,<DE ARGUMENT>]

The function is called (This should appear as one line with a space after "<C ARGUMENT>.") by passing the function number as the first argument to the function call. The function numbers are listed in the back of most CP/M documentation manuals. The value passed as the first argument will be "inserted" in the C register when the system call is made.

Many system calls require a second argument which is passed to the DE register pair. This argument is optional to the CPMFN, but if the system call requires a second argument, you must include this value in the CPMFN call.

When the function returns, the variable set equal to the function call (T in the example) will be equal to the value of the HL register pair.

The CREATE statement now creates a dummy file the size of the size variable, if it is specified. This means that the space specified in the CREATE statement is reserved on the disk when the CREATE statement is executed.

baZic now updates the file directory when a program is exited. Previously, under some conditions, the file directory was not updated, leading to errors in the files.

baZic files can now be CLOSEd without previously OPENing the file.

The implementation of the cursor positioning and clear screen statements has been changed at the beginning of baZic to allow more CRTs to be configured. Two modes are now supported for setting the cursor addressing and clear screen sequence. At the beginning of the cursor addressing and clear screen sequence is a byte which controls the mode. If the byte contains a 0FFH, baZic will assume a machine language routine is in place starting at the next location after the mode byte and will jump to this location and execute the code located there.

If the mode byte is any other value, baZic will assume that the byte represents the number of codes to output and will output that number of bytes, starting with the first byte following the mode byte. The string of codes in this mode must end with a "$" code (36 Decimal or 24 Hex). The SOROC option of the CRT program uses this mode while all other terminals in the CRT program use the first mode.

The HAZELTINE option uses the first mode described but is slightly different in that the HAZELTINE requires the cursor addressing codes be issued column,row whereas most CRTs require a row,column sequence. This sequence is changed for the HAZELTINE option by reversing the bytes at location 0131H and 0138H. The MOV E,L instruction at 0131H is changed to a MOV E,H and the MOV E,H at 0138H is changed to a MOV 3,L. This reverses the order the cursor addressing codes are output.

The CRT program must be run on a "virgin" copy of baZic as delivered on the master diskette. You cannot configure a copy of baZic for the SOROC and then run CRT again to configure the same copy for a Z19. You must use a copy from the master diskette to configure for each different CRT you might have.

Datapoint CP/M-80
Version 2.21

Version 2.21 of Datapoint 1550 CP/M provides the following fixes and enhancements over version 2.20.

1. Datapoint printers with hardware handshaking are now supported. Choose the HARDWARE HANDSHAKING option in CONFIG as well as selection of 9600 baud for port 'B'. Plug your Datapoint supplied cable into comm port 2, which is the lower socket on the back of your machine. This should fix any problem you've been having with your printer.

   The hardware handshaking option checks three signals on the comm port. CTS (pin 5) and DSR (pin 6) are level-activated handshaking pins; that is, if either or both of these lines are low, transmission ceases. DCD (pin 12) is a pulse-activated handshaking signal. When a character is received by the printer a DCD pulse is sent back once the character is accepted. When the buffer is full, an acknowledgement is not sent back until there is enough room in the buffer

(continued next page)

to accommodate further characters. It should be noted that while CTS and DSR must be high for any characters to be sent out, the user need not supply the DCD signal for correct operation.

If the user employs a device that does not provide hardware handshaking he must jumper pins 5 and 20 on the machine since the Datapoint hardware will not allow transmission of any characters without a high level on the CTS pin.

All the techniques described above apply equally to comm port 1; however, the user should note that because of its hardware characteristics, in addition to all of the above considerations, the user must supply both the XMT clock (pin 15) and the receive clock (pin 17) at 16 times the desired baud rate.

2. The INSERT/DELETE Line functions of the ADM-31 emulator caused the system to crash if these functions were performed on the last line. This bug was fixed.

3. The inverse video functions of the ADM-31 emulator were set to <ESC> 'A', and <EDC> 'B' to respectively turn on and off the inverse video function. The present version changes these sequences to <ESC> ')' and <ESC> '(' which are those used on the ADM-31.

4. Certain often used keys in CP/M were not very accessible to the user, thus they were moved to more convenient locations. The 'box' key on the main keypad (second row from top, all the way to the right) was changed so that its unshifted value produces a TAB (↑I) and its shifted value produces a 'ˋ'. The margin release key (main keyboard, top row, second from the right) produces a DEL (7F hex) code in the non-shifted mode, while the shifted key produces a backslash ('\').

5. The interrupt service routines, the routines used to handle such things as reading the keyboard and displaying characters on the CRT, employed a technique that would not allow certain programs to run. An example of such programs were programs compiled with PASCAL/Z. We are now employing a different technique in these routines that fixes these problems.

6. The cursor has been changed to simply display the character under it as the inverse (black on white becomes white on black) of the character in the cursor position. This makes the cursor easier to see and allows the user to see what's under the cursor.

7. A number of users have had some problems installing our two data communications programs BSTAM and BSTMS on their Datapoint machines. After some experimentation we can recommend the following installation procedure.

   A. Use Comm port 2. The cable you use must either provide CTS or you must jumper pins 5-20 on your Datapoint.

   B. Assemble the UDATAPT.ASM file supplied on your BSTAM or BSTMS distribution disk.

   C. Install the resulting UDATAPT.HEX file in your communications program(s) as per the instruction in the manual.

   D. Using CONFIG, set port 'B' to 'NO PROTOCOL'

   E. Also using CONFIG, set the BAUD RATE to the same rate as the machine you wish to communicate with.

SELECTOR IV
Version 2.17

Any line, page or batch definition prior to Version 2.15 will not load with this version and must be redefined. GLector IV must be upgraded to Version 2.15 to interface with this version's main menu.

All .DEF files as well as the DEMO.-PAG (page report demonstration), have been initially defined expecting all .DEF, .DAT and .KEY files to be on drive A. If they are PIPped to another drive the definitions must be modified.

The internal name of '.DEF' files is no longer used, so that if the '.DAT', '.KEY', and '.DEF' files are all renamed with the same new name prefix, the new name will be recognized.

The system may now be run from any drive. Erase any existing TERM.DAT and re-implement the system parameters. The only parameter actually changed is the operating system type (CP/M...) which is suffixed now with the program drive letter (e.g. "2E" vs. "2").

Be sure to include the drive declaration when entering the name of a report, batch, etc. definition file after a merge operation, otherwise, a "not a correct definition" error message may be generated.

Subtracting a date from a date should return the number of days for dates within 89 years of each other. Remember to pass the result to a field configured for integer numbers vs. a date field.

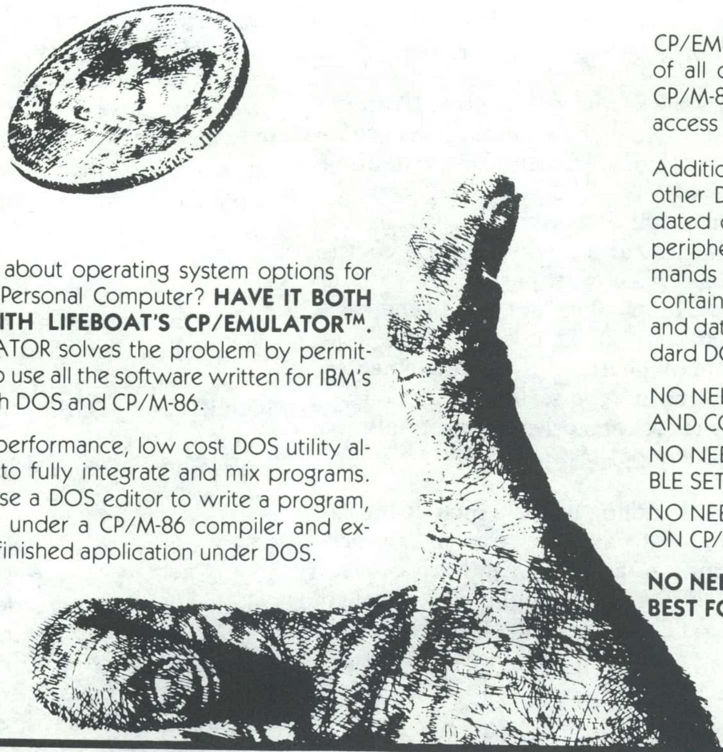
Note the use of the dummy definition (variable def) used with the DEMO.-PAG page report definition. Such dummy definitions may have any prefix filename, contain a variety of fields to be used to accumulate or receive derived data, and may be used with either a batch update definition or any line or page report definition. The supplied DEMO.PAG produces a multi-paged pseudo statement output as a definition example.

Data entry editing features have been modified somewhat. Only '<' and '>' will move the cursor up and down field by field. The use of ',' and '.' have been discontinued. This allows entering numbers beginning with a period.

When anywhere within a field, a delete or back-space will print underscores from the present cursor position to the end of the field. A '?' will retype the field as its value was upon initial entry and place the cursor on the first character. Note that no characters to the right of the cursor are entered into the field when 'RETURN' is pressed. A '?' retype followed by a 'RETURN' nulls the field. All field characters remain on the screen until over-typed.

When calling any function from any menu, if the called program is not on the operating disk, a message will be displayed prompting the user to take out the current program disk and substitute it with one containing the appropriate '.INT' program file.

IBM'S DOS or CP/M-86?™



Confused about operating system options for your IBM Personal Computer? **HAVE IT BOTH WAYS WITH LIFEBOAT'S CP/EMULATOR™.** CP/EMULATOR solves the problem by permitting you to use all the software written for IBM's PC for both DOS and CP/M-86.

This high performance, low cost DOS utility allows you to fully integrate and mix programs. You can use a DOS editor to write a program, compile it under a CP/M-86 compiler and execute the finished application under DOS.

CP/EMULATOR extends the scope and capacity of all of your software. With CP/EMULATOR, CP/M-86 programs run quicker with faster file access than with CP/M-86 itself.

Additionally, your program will enjoy all the other DOS advantages, such as large file size, dated directory displays, and more. The DOS peripherals are already installed and DOS commands are fully available. The package even contains a utility program to transfer programs and data files from a CP/M-86 diskette to a standard DOS diskette.

NO NEED TO LEARN A NEW SET OF UTILITIES AND COMMANDS;

NO NEED TO END UP WITH TWO INCOMPATIBLE SETS OF DISKETTES;

NO NEED TO SPEND HUNDREDS OF DOLLARS ON CP/M-86, WHEN CP/EMULATOR COSTS $75.

NO NEED TO WONDER WHICH OPTION IS BEST FOR YOUR IBM PERSONAL COMPUTER.

PMATE™: Editor-in-Chief

Perform miracles of manipulation on your keyboard with Lifeboat's PMATE.

This new generation text editor is the most sophisticated text editor available today and is bristling with features previously unavailable on microcomputers, making it ideal for virtually any program or data file editing.

PMATE's command set includes full screen single keystroke editing, horizontal scrolling, automatic disk buffering, macro command language, text formatting, expression evaluation, conditional branching, I/O with prompting, and other programming language constructs. PMATE makes use of 11 buffers for storage, and includes commands permitting work on more than one text at a time. Unique to PMATE is the facility for user customization. Keystroke functions can be redefined, and sequences can be programmed to directly execute macros. Video commands can be changed, and macro functions can be written, to emulate any other editor with which you may be familiar. PMATE provides full side-scrolling, and can be used with virtually ANY video terminal on the market. IF you use an editor, you need PMATE.

PMATE is the **only** text editor you'll ever need.

PMATE-86 is available for IBM's Personal Computer DOS, SB-86™ and MS-DOS™. Also available is PMATE for SB-80 and other CP/M-80-compatible operating systems.

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

For more information, send to:

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

SB-80 and SB-86 are trademarks of Lifeboat Associates
PMATE and PMATE-86 are trademarks of Phoenix Software Asso. Ltd.
MS-DOS is a trademark of Microsoft, Inc.
CP/EMULATOR is a trademark of Lifeboat Associates.
CP/M-80 and CP/M-86 are registered trademarks of Digital Research, Inc.
This ad was designed by DocuSet™.
Copyright © 1981, by Lifeboat Associates

LIFEBOAT HAS THE ANSWER

New Products

The products described below are available from their authors, computer stores, software distributors and publishers.

Cache/Q
Queue Computer Corporation

This product is designed to enhance the speed of a CP/M-80 system by a factor as high as 35 times. Cache/Q requires a CP/M-80 2.2 system with 64k bytes of memory or less; it can take advantage of bank-selected memory. Transfers to and from the disk drives are buffered, reducing the amount of disk activity required for a given application. Cache/Q is transparent to user and system programs.

This product features an interactive installation program, a reconfiguration program through which a user may specify which files or class of files are to be buffered, and a display program showing the operating statistics of Cache/Q.

Comet-FORTH
Computer Methods

This product is designed only for use on Cromemco computers. Most primitives and some high level words are written in assembler, for speed of execution. It is interactive, and Z80 assembler code may be mixed with high level Comet-FORTH code. It occupies about 8k of memory, and is a standalone operating system. Memory space is reconfigurable.

A line editor and a full screen editor are provided with the product. Comet-FORTH is compliant with the FORTH 79 standard; it is fully interrupt-driven, to allow type-ahead, overlap of computation and I/O.

A multi-user version is called Comet Multi-FORTH and permits up to eight on-line terminals and an additional number of non-terminal tasks. Each user is always memory resident and has 32k of dedicated memory. The FORTH dictionary, assembler vocabulary, and screen editor do not require user memory. All users have access to the system resources.

Graftalk
REDDING Group Inc.

This interactive graphics product is intended for the business user, and is oriented towards the creation of bar charts, exploded pie charts, and other line, point and symbol plots. A joystick and digitization can be added. Color is also fully supported.

Graftalk features more than 100 commands, including multi-level commands and windowing on the plotting surface. Bar charts can be vertical, horizontal, in reverse, shaded, in color; axes can be labelled, and curves added. Likewise pie charts can be shaded, in color; labelling of the various elements is possible. Line and symbol charts can be implemented with automatic scaling. These different chart categories can be combined into composites.

Text handling features include the capability to adjust character size, perform paragraphing, and erase selectively. A compatible editor called SCATE can be embedded to permit easy text manipulation, without leaving Graftalk.

Advanced users can employ declared variables, separate axes, boxes, grids, absolute and relative moves, three levels of coordinate systems, windows, viewports and digitizing.

The authors of Graftalk state that the system requirements for this initial release are more restrictive than they will be in the future. Graftalk requires 48k TPA, recommended for use with graphic CRTs and plotters as primary graphics devices; only printers which can be dumped to from the CRT can be used (Diablo, Epson or Anadex with graphics). Graftalk runs on 8251 or SIO computers, requires CP/M-80 (2.x or above), MP/M or compatible operating system. ADM-3A and 3A+ retrofit CRT's as well as Televideo retrofit CRT's are supported. Plotters supported follow: Tektronix 4662 with or without option 31; Hewlett-Packard 7220, 7221, 7225; Houston Instruments DMP3, DMP7.

MAGSORT
Micro Applications Group

This SORT/MERGE/SELECT utility requires no dedicated memory and is easily callable from high level languages. It may be called from CBASIC2, Microsoft BASIC Interpreter, Microsoft BASIC Compiler, FORTRAN-80, Pascal/MT+ and PL/I-80. A self-relocating bootstrap routine writes the user program to disk, then loads MAGSORT into memory; so the entire memory is available for sorting, a feature designed to speed sorting. When sorting is completed, MAGSORT reloads the user program into memory and returns with a status code. MAGSORT may also be run standalone.

File size is defined by the operating system and work space available; record size is also limited only by available memory. Sorting is on up to ten keys, ascending and descending independently. Records may be selected or excluded with up to four independent keys. Select/Exclude comparisons permit <, >, and =; wild card characters are permitted in Select/Exclude keys, and lower case letters may be treated as upper case.

Merge allows two files to be merged in sorted order, or appended to one another in existing order. Work and output disks may be changed during processing. Up to the first 255 records in a file may be skipped. Parameters may be passed in a string from the user program, or from a parameter file; sorts can also be performed interactively.

No special interfacing is required for use with the host languages, and no relocation is needed for different memory configurations. MAGSORT is written in 8080 assembler and BASIC, requires CP/M-80 1.4 or 2.2, or CP/M or MP/M II; it supports 8080/Z80 or 8085 CPU's. A minimum disk capacity of 50k per drive is required, and a total disk capacity of 50k is needed.

SB-80 Disk Operating System
Lifeboat Associates

SB-80 is designed to run on a broad range of 8080/Z80-based systems, using floppy disks, hard disks or combinations of the types. It is fully compatible with CP/M-80 and **available only to OEM's.**

SB-80 allows use of files larger by a factor of 16, includes an integral background printing capability and is designed to have a more flexible set of intrinsic commands than does CP/M-80. Fundamental utilities are provided, including a file-transfer handler, text editor, assembler and debugger.

SB-80 is intended for 8080/8085/Z80-based microcomputers with a minimum of 20 kilobytes of continuous read/write memory starting at address 0. (Where ROM exists in low memory, the Lifeboat Associates MMU board can remap memory and optionally add 16 additional kilobytes of RAM; it is for Z80 systems only.) S-100 (IEEE 696) systems, single-board computers and desktop systems can be suitable SB-80 environments.

Many controllers for floppy, minifloppy and/or hard disks can be used. Where a CP/M-80 BIOS for the controller is available in source code, it can be adapted for SB-80 use. Where a BIOS must be generated, SB-80 documentation provides sufficient information. Four separate I/O devices other than disk controllers are available through the system at any time, selected from among as many as 16, depending on the BIOS implementation.

The software that constitutes the operating system itself is in three parts: the Disk Operating Subsystem (DOS), SB-80's file-manager, resides on reserved system tracks of a bootable disk; the Basic Input/Output Subsystem (BIOS), the hardware-specific segment, also resides on reserved system tracks; the Command-Line Interpreter (CLI), which executes intrinsic commands and loads user programs and initiates their execution is held on disk as a file.

Including page zero (addresses 0000-00FFH), the operating system requires at least 10 kilobytes of memory. This figure increases when the hardware-dependent BIOS must be enlarged, as for the inclusion of extra I/O features. Excluding page zero, the address at which the system resides in memory can be determined by the user.

Operating system facilities allow you to create, delete, rename, read and write both sequential files with variable-length records and random-access files of fixed record length. As many as 16 disk drives are supported, and file space is dynamically allocated to maximize disk utilization. The maximum possible size of a file under SB-80 is 128 megabytes. This 128 Mb is also the maximum capacity of a single drive, and is dynamically allocated among files as needed.

SB-80's resident background printing function eliminates the disadvantages of despooling software not integral to the operating system. Background printing of files is queued, with as many as six files in the queue at one time. The operator can suspend, resume or cancel the background printing function, add files to the queue or remove them from it at any time. Batch processing is also integral to the operating system.

New Publications

Don't
(Or How to Care For Your Computer)
Rodney Zaks
This book describes the proper way to treat your computer and peripherals, including media. Chapters give you pointers on planning a computer room, preserving and retaining system documentation, handling security for hardware and data; preventive maintenance is also described.

How To Buy The Right Small Business Computer System
C. Roger Smolin
This book attempts to answer the questions prospective micro owners should ask before buying a computer for their office. It guides the reader on the basics of how a computer operates, what to expect from it, how to shop for and purchase it. In addition, the businessman is instructed on preparing for the new computer, and installing it.

How To Get A Free Desktop Computer
Vernon K. Jacobs
This book is not exactly what the title implies, but rather another guide to purchasing a small computer system. However, in this publication, the importance of good software is stressed as a prerequisite for an efficient, cost-effective system. Mr. Jacobs concentrates on the financial analysts and planners, and their particular needs in a personal computer and its software.

The Devil's DP Dictionary
This is a side-splitting spoof of the computer trade's jargon, spoofing the sometimes ridiculous expressions we use. It's a good book to relax and laugh with.

Operating Systems

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

Hard Disk Modules

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

ZAP80

(*Editor's Note:* This product brief of ZAP80, from Phase Four Systems, Inc., was written by the author of ZAP80, Mark Rollins.)

ZAP80 is an extensive, menu-driven disk access utility for SB-80; CP/M-80, and other compatible 8080/Z80 operating systems. It was originally written to allow direct access to the disk surface by specifying a track and sector number for any sector. In its development, it was first expanded to include extensive file utility servicing to access and patch file sectors, as well. Then, several additional functions were added, such as direct memory display, string search, file compare, track initializing, and chaining.

There are four menus which display the 51 commands available in ZAP80 to perform the above functions. In addition, there is a self-contained Configure System command which allows the user to install the cursor up, clear screen, and cursor addressability terminal controls used by ZAP. Of the three, only the cursor up is required by ZAP for proper screen displays; if the other two are not provided, ZAP will use its own internal routines.

ZAP80 simultaneously maintains two sets of parameters for accessing the disk: file parameters and logon parameters. The latter are called logon because they refer to the current logged-on drive, and include the commands Select Drive, Set Track, and Set Sector. The file parameters allow access to any sector of a specified file, while the logon parameters allow direct access to the disk surface by specifying a track and sector on the logged-on drive. The file does not have to be on the logged-on drive, and can be left open while a logon sector is accessed.

For file operations, ZAP80 maintains an internal file control block, and all file accesses are performed through standard DOS function calls. Random reads are made by the user specifying either a relative address or a logical sector number, which ZAP internally translates to the proper file extent and relative sector number within the extent, before making the function call.

For file reads, ZAP80 expects the BIOS to perform a set track and set sector during the read operation. If a BIOS does not do so, the resulting file parameter display of the track and sector values will be arbitrary. It should be noted, however, that the track and sector display here is a convenience (allowing, for instance, subsequent access to a file sector by using the logon parameters) and does not materially affect actual file i/o operations.

For logon parameter accesses, ZAP80 specifically recognizes the disk parameter headers of SB-80 2.5+, CP/M 2.0+, and DMADOS 8.0+; all others default to CP/M 1.4. For CP/M 1.4, ZAP uses its own internal parameters, which are user modifiable. There are currently provisions for two different drive environments, initially set to 'standard' 8-inch single density for the first, and double density (52 sectors/track) for the second. The user can then toggle between the two to set the current logon environment.

Included in ZAP's internal parameter area are the sector translation tables for the two drive environments. Let's look at a sample of how ZAP works by discussing two demonstrations: (1) patching one of these translation tables, and (2) 'unerasing' a deleted file.

Using ZAP to patch a translation table

When ZAP is run, the translation tables are located in memory beginning at 1a0h for the first table, and 1c0h for the second. We shall work with the first table, which is initially setup for an 8-inch, single density drive with 'standard' skew (i.e., if your BIOS is set up for 'normal' CP/M 1.4 standards, when you ask to read 'logical' sectors 1, 2, 3, 4, etc. in that order, you will actually be reading sectors 1, 7, 13, 19, etc. This is called 'skewing'.).

Let's assume, for our purposes, that your BIOS in fact does not need this skew to be known by the outside world (that is, external programs), and that you don't want the table to read 1, 7, 0d (which is hex for 13), 13 (hex for 19), etc. Here's how you proceed:

Type on the command line at the console (before performing this demonstration, be sure to use the CS (Configure System) command to at least put in the cursor up terminal control, otherwise the display will not match the following discussion):

```
ZAP ZAP.COM
```

After ZAP signs on, type a carriage return. This will display the first sector of ZAP. Note that the address displayed at the left side of the screen is 0100, even though it is (relative) address 0000 in the file. This is because ZAP maintains a File Base which is initially set to 0100 to account for the fact that this is where programs are normally loaded in memory. If you have a program which originates at a different location in memory, you can use the FB (File Base) command to have the display match an assembly listing (for instance).

Now type a second carriage return, which will display the second sector. The display should look like fig. 1 in this article. To patch the first table (recall it is at location 1a0h), use the HP (Hex Patch) command; type

```
HP 1a0
```

You will see the beginning of the sector blanked out up to 1a0, and the cursor positioned at the byte at 1a0, after which you type 01, followed by 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1a. At this point, you end the patch by typing <escape>, and then write it back to disk by using the FW (File Write) command.

The sector is now patched with the new table. To actually use it, you must first exit ZAP, and then reenter, since it has been patched on disk, but not in memory in the running program.

'Unerasing' a deleted file

Let's say you have accidentally deleted (ERAsed) a file. You can use ZAP to unerase it. Here's how it is done: On the command line at your terminal, type

```
ZAP
```

ZAP80 signs on with the logon parameters preset to the first sector of the directory (normally, track 2, sector 1). Use the RL (Read Logon) command to read the first sector. The display should look something like fig. 2 in this article (assuming it is the very first file that was deleted).

As you will notice, each sector of the directory has the directory control blocks, called DCB's (you will usually see them referenced as FCB's in the CP/M documentation; the difference is that the FCB is the control block used when actually performing the i/o operation and is either 33 or 36 bytes long, depending on whether the operation is sequential or (for CP/M 2.0+) random, where the DCB is the first 32 bytes of an FCB as it is actually written to the directory), for four directory entries. They may be single-extent entries for four files, or multiple-extent entries for less than four files.

In either case, the operating system places 0e5h in the first byte of the DCB to signify that the file has been erased and the directory entry is free for use by another file. In order to 'unerase' the first file, type

HP

followed by a carriage return (this will position you at the first byte of the sector), and then type

00 (← this assumes user 0)

which undeletes the file. Then use the WL (Write Logon) command to write the sector back to disk. This command prompts you to make sure that it is the logon sector you want to write. After you respond 'Y', the sector will be written back, and the file will reappear in the directory (note: make sure you unerase ALL of the directory entries for the file).

By using the above procedures, files can be edited (much faster than using an editor which writes the entire file back to disk, though insertions are not possible), sectors or directory entries can be reconstructed, and files can be compared (ZAP displays the sectors which have differences) and then patched to make them the same. And these are only a few of the powerful features of this new utility.

```
COMMAND==>


file  parameters==>    track  4  sector  2     file= B:ZAP.COM  LSN=001

logon parameters==>    track  2  sector  1     drive B     dens=64(o/s)

         00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   0123456789ABCDEF
         -----------------------------------------------   ----------------
0180 |  02 1B 3D 00 00 00 00 00 00 00 00 00 00 00 00 00   ..=.............
0190 |  00 00 00 00 00 00 00 00 00 20 20 00 00 00 00 00   .........  .....
01A0 |  01 07 0D 13 19 05 0B 11 17 03 09 0F 15 02 08 0E   ................
01B0 |  14 1A 06 0C 12 18 04 0A 10 16 00 00 00 00 00 00   ................
01C0 |  01 02 07 08 0D 0E 13 14 19 1A 1F 20 25 26 2B 2C   ........... %&+,
01D0 |  31 32 03 04 09 0A 0F 10 15 16 1B 1C 21 22 27 28   12..........!"´(
01E0 |  2D 2E 33 34 05 06 0B 0C 11 12 17 18 1D 1E 23 24   -.34..........#$
01F0 |  29 2A 2F 30 00 00 00 00 00 00 00 00 00 00 00 00   )*/0............

COMMAND==>RL


file  parameters==>    track  4  sector  2     file= B:ZAP.COM  LSN=001

logon parameters==>    track  2  sector  1     drive B     dens=64(o/s)

         00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   0123456789ABCDEF
         -----------------------------------------------   ----------------
  00 |  E5 53 59 53 54 45 4D 20 20 43 4C 49 00 00 00 20   .SYSTEM  CLI...
  10 |  02 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
  20 |  00 58 44 53 20 20 20 20 20 43 4F 4D 00 00 00 07   .XDS     COM....
  30 |  04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
  40 |  00 58 44 20 20 20 20 20 20 43 4F 4D 00 00 00 0D   .XD      COM....
  50 |  05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
  60 |  00 58 44 46 20 20 20 20 20 43 4F 4D 00 00 00 0A   .XDF     COM....
  70 |  06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

COMMAND==>
```

(continued from page 2)

OUTPUT commands allow a byte to be sent to/from a specified I/O port at the command line level while in DEBUG.

The standard SB-86 programs (e.g. DEBUG, COMMAND, EDLIN) all automatically treat the the last command as the template for the next. Characters may be edited at any point in a line without having to retype the remainder of the line to the right of the cursor, etc.

An interesting safeguard is the use of a duplicate directory with a read occurring after every write to the directory(s). Those using single disk systems and those who prefer to store data on separate diskettes will be pleased to learn that there is no need to log in disks by typing a control-C. Also there is no physical file/disk size limitation, unlike many operating systems which are restricted to eight megabytes. That is, SB-86 doesn't do silly things like breaking a twenty-four megabyte disk into three logical drives.

There's no overhead for non-128-byte sectors either, which is particularly nice when different physical sectors are to be supported.

So in summary, it's easy to see why IBM chose SB-86, since among other things, it's written entirely in 8086 assembly language, and not some kludge which was translated from an 8-bit operating system. The file structure is fast and efficient since "extents" are not employed; therefore access to the directory track is minimized.

And on, and on, and on ... Take a look at the IBM DOS manual if you want to the way operating system documentation should be prepared. You'll recognize it immediately. It's the one without the copyright notice printed on every page.

Those of you interested in sixteen-bit microprocessor architecture should take a look at "16-Bit Microprocessors", a paperback published by Howard W. Sams & Company, Incorporated. Whether hardware is of great interest or not you should find the benchmarks provided for the 8086, Z8001 and Z8002, LSI-11, TI9900, 68000 and the NS1600 fascinating. Bubble sort, ASCII string search, multiplication/division and lookup table algorithms are provided for each microprocessor and the corresponding execution times allow for an interesting comparison of these widely varying architectures.

By the way, don't get the idea that you are going to be left behind in the rush to jump on the 8088/8086 bandwagon. Those of you with S100 systems will be pleased to learn that a number of manufacturers are offering S100 cards with 8088 or 8086 processors which will run SB-86. One of these offers both 8085 and 8088 processors so that you can run either 8- or 16-bit software anytime you wish.

Furthermore, most of your favorite programs are being transported to these 16-bit environments. And, as mentioned earlier, utilities available allow you to move your assembly language programs easily. And of course anything written in Microsoft high level languages is a piece of cake. Just recompile and away you go ... PMATE is undoubtedly the best available text editor for micros and is already running on the IBM Personal Computer as well as other 8088/8086 machines.

TELECOMMUNICATIONS

More of you should be exploiting the telecommunications capabilities of micros, whether for accessing your system remotely or the systems currently available to you, many of which offer programs and services at no charge.

The modem world has long been dominated by the PMMI board for S100 machines, but until recently no comparable counterpart has been available for non-S100 machines. However, Hayes has recently announced the "SmartModem", which among other things is a very "Smart Idea". This unit connects to the RS232 port of your microcomputer and offers tone and rotary auto-dialing as well as auto-answer. Also provided is a speaker to permit the user to listen in. Status information is provided by the integral LEDS which indicate current operating mode, auto-answer, carrier detect, off hook, receive data, send data, terminal ready and modem ready. While use of this unit is restricted to 300 Baud, it's clearly a nice design.

Those of you who have not gotten into the "CBBS" world (Computerized Bulletin Board System) should consider investigating this free source of classified ads and public domain software. Perhaps one of the most sophisticated of such systems in the country is that of Kelly Smith (805-527-9321) who has twenty megabytes of programs on line for the taking. Just dial up, browse around and pick up whatever is of interest. Most systems have a Telephone Directory online which you can transfer to your system and use to contact any of the CBBSs in the country. It's reassuring to know that there are people who care enough about microcomputers to provide the kind of service Kelly offers at no charge to the users.

These systems require a considerable amount of maintenance and users are often more demanding than one would expect paying customers to be. Recently Kelly mentioned that one of the drives had developed bearing problems. He decided that he would just let it grind on and fix it when he got around to it. Unfortunately, the grinding produced metal filings which found their way to the surface of the disk and along came the head and flew right into a metallic mountain. Such things don't contribute much to aerodynamic stability.

Kelly gave up a weekend and rebuilt it right on the spot, so it was soon available for use. In the meantime some users were irate at suffering the downtime. The moral of that story is when your disk speaks ... listen ...

Ward Christensen started the underground micro-communications movement, which has grown by leaps and bounds, with his now legendary program MODEM. As you many of you are aware, Ward has made a lot of invaluable contributions to the microcomputer world in terms of communications programs and protocols, CBBS® s, disassemblers, disk patching programs, standards for documentation of microcomputer source code and much, much more.

BELL, BOOKS, BIBLIOGRAPHIES AND DATA BASES

Those of you who are interested in UNIX will find Bell Laboratories' documentation most informative. In particular, pick up a copy of "DOCUMENTS FOR USE WITH THE UNIX TIME-SHARING SYSTEM" available from Bell. This document includes, among other things, sections called "UNIX for Beginners" and "UNIX Summary" respectively which give a good overview of the operating system and its many features. UNIX for Beginners is authored by our old friend Brian W. Kernighan and contains an annotated bibliography.

Speaking of bibliographies, be sure that you take a look at Douglas R. Hofstadter's book "Godel, Escher and Bach an Eternal Golden Braid". This fine example of scholarly work contains a fascinating annotated bibliography. Hofstadter is currently the author of the "Mathematical Games" column in *Scientific American*.

If you are looking for information on 8086 assembly language don't miss "The 8086 Book" by Russell Rector and George Alexy. This McGraw-Hill publication covers all aspects of the 8086 instruction set and is filled with illustrative examples.

If data base management is of interest to you take a look at the November 17, 1981 issue of Electronics, page 129. There's an interesting article on Micro Data Base Systems (MDBS). Sequential, direct, ISAM, CODASYL and MDBS are discussed.

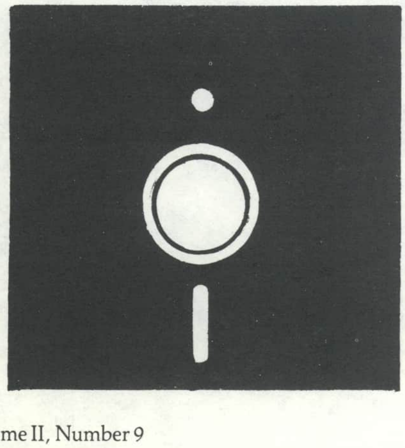
There is an interesting book which has just been published by Addison-Wesley Publishing Company called "Real Time Programming - Neglected Topics" by Caxton C. Foster.

The author presents an excellent introduction to the topic of interrupts; both simple and hierarchical and semaphores are also treated in an enlightening manner. This text is based upon a course at the University of Massachusetts called "Real Time Programming" and includes a number of interesting hardware examples. This book contains a number of topics which are not discussed in simple terms elsewhere, ranging from The Sampling Theorem to communication over restricted pathways.

* * *

Your response to the editorials of recent months has been most encouraging and many of the topics suggested for discussion will be covered in future editorials, so keep those cards and letters coming!

Edward H. Currie



VERSION LIST

January 13, 1982

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

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

S Standard Version
M Modified Version
P Processor
MR Memory Required

Product	S	M	P	MR	
ACCESS-80	1.0		8080/Z80	54K	
Accounts Payable/Cybernetics	3.1		Z80	64K	Needs RM/COBOL
Accounts Payable/MC	1.0		8080/Z80	56K	For CP/M 2.2
Accounts Payable/Structured Sys	1.3B		8080	52K	w/It Works run time pkg.
Accounts Payable/Peachtree	07-13-80			48K	Needs BASIC-80 4.51
Accounting Plus			8080/Z80	64K	
Accounts Receivable/Cybernetics	3.1		Z80	64K	Needs RM/COBOL
Accounts Receivable/MC	1.0		8080/Z80	56K	CP/M 2.2
Accounts Receivable/Peachtree	07-13-80		8080	48K	Needs BASIC-80 4.51
Accounts Receivable/Structured Sys	1.4C		8080	56K	w/It Works run time pkg.
Address Management System	1.0		8080		Requires 2 drives
ALDS TRSDOS		3.40	8080	32K	Needs TRSDOS. TRSDOS Macro-80
ALGOL 60	4.8C		8080	24K	
ANALYST	2.0		8080	52K	Needs CBASIC2,QSORT/ULTRASORT
APL/V80	3.2		Z80	48K	Needs APL terminal
Apartment Management (Cornwall)	1.0	1.0	8080		Needs CBASIC2
ASM/XITAN	3.11		Z80		
Automated Patient History	1.2		8080	48K	
BASIC Compiler	5.3	5.3	8080	48K	
BASIC-80 Interpreter	5.21	5.21	8080	40K	w/Vers. 4.51,5.21
BASIC Utility Disk	2.0	2.0	8080	48K	
BOSS Financial Accounting System	1.08		8080	48K	Needs 2/3- drives w/min 200k each, & 132-col. printer
BOSS Demo	1.08		8080	48K	
BSTAM Communication System	4.5	4.5	8080	32K	
BDS C Compiler	**1.45**	**1.45T**	**8080**	**32K**	**w/'C' book**
Whitesmiths' C Compiler	2.0		8080	60K	
BSTMS	1.2	1.2	8080	24K	
BUG / uBUG Debuggers	2.03		Z80	24K	
CBASIC2 Compiler	2.08		8080	32K	w/CRUN(2,204P, & 238)
CBS Applications Builder	1.3		8080	48K	Needs no support language
CIS COBOL Compiler	4.4,1		8080	48K	
CIS COBOL Compact	3.46	3.46	8080	32K	
FORMS 1 CIS COBOL Form Generator	1.06	1.06	8080		
FORMS 2 CIS COBOL Form Generator	1.1,6a		8080		
Interface for Mits Q70 Printer					CP/M 1.41 or 2.XX
COBOL-80 Compiler	4.01	4.01	8080	48K	
COBOL-80 PLUS M/SORT	4.01		8080	48K	
CONDOR II	2.06		8080	48K	
CREAM (Real Estate Acct'ng)	2.3		8080	64K	CBASIC needed
Crosstalk	1.4		Z80		
DATASTAR Information Manager	1.101		8080	48K	
Datebook-II	2.03		8080	48K	Needs 80x24 terminal
dBASE-II	2.02A		8080	48K	
dBASE-II Demo	2.02A		8080	48K	
Dental Management System 8000	8.7A		8080	48K	Needs CBASIC
Dental Management System 9000	1.07		8080	48K	Needs CBASIC
DESPOOL Print Spooler	2.1A		8080		
DISILOG Z80 Disassembler	4.0	4.0	Z80		Zilog mnemonics
DISTEL Z80/8080 Disassembler	4.0		8080/Z80		Intel mnemonics,TDL extensions
Documate/Plus	**1.4**		**8080**	**36K**	
EDIT Text Editor	2.06		Z80		
EDIT-80 Text Editor	2.02	2.02	8080		
FABS-I	2.6		8080	32K	
FABS II	**4.11**		**8080/Z80**	**48K**	
FILETRAN	1.20			32K	1-way TRS-80 Mod I,TRSDOS to Mod I CP/M
FILETRAN	1.4			32K	Needs TRSDOS. 2-way TRS-80 Mod I,TRSDOS & Mod I CP/M
FILETRAN	1.5			32K	1-way TRS-80 Mod II,TRSDOS to Mod II CP/M
Financial Modeling System	2.0			48K	
Floating Point FORTH	2		8080/Z80	28K	
Floating Point FORTH	3		8080/Z80	28K	
FORTRAN-80 Compiler	3.43	3.43	8080	36K	
FPL 56K Vers.	2.6		8080	56K	
FPL 48K Vers.	2.6		8080	48K	
General Ledger/Cybernetics	1.3C		Z80	48K	Needs RM/COBOL
General Ledger/MC	1.0		8080/Z80	56K	Needs CP/M 2.2 or MP/M
General Ledger/Peachtree	07-13-80		8080	48K	Needs BASIC-80 4.51
General Ledger/Structured Sys	1.4C		8080	52K	w/It Works Package

VERSION LIST

Product	S	M	P	MR	
General Ledger II/CPaids	1.1		8080	48K	Needs BASIC-80 4.51
GLECTOR Accounting System	2.02		8080	56K	Use w/CBASIC2,Selector III
GLECTOR IV Accounting System	1.0		8080		Needs Selector IV
HDBS	1.05A		+	52K	
IBM/CPM	1.1		8080		
Insurance Agency System 9000	1.06		8080		Needs CBASIC
Integrated Acctg Sys/Gen'l Ledger			8080	48K	Needed for 3 pkgs. below
Integrated Acctg Sys/Accts Pyble			8080	48K	
Integrated Acctg Sys/Accts Rcvble			8080	48K	
Integrated Acctg Sys/Payroll			8080	48K	
Interchange			Z80	32K	
Inventory/MicroConsultants	5.3		8080/Z80	56K	Needs CP/M 2.2
Inventory/Peachtree	07-13-80		8080	48K	Needs BASIC-80 4.51
Inventory/Structured Sys	1.0C		8080	52K	w/It Works Package
Job Cost Control System/MC	1.0		8080/Z80	56K	Requires CP/M 2.2
JRT Pascal System	1.4		8080	56K	
LETTERIGHT Text Editor	1.1B		8080	52K	
LINKER			Z80		
MAC	2.0A		8080	20K	
MACRO-80 Macro Assembler Package	3.43	3.43	8080/Z80		
Magic Typewriter	3		Z80	48K	
Magic Wand	1.11		8080	32K	
MAGSAM III	4.2		8080	32K	
MAGSAM IV	1.1		8080	32K	Needs CBASIC
MAILING ADDRESS Mail List System	07-13-80		8080	48K	
Mail-Merge	3.0		8080		
Master Tax	1.0-80		8080	48K	
Matchmaker			8080	32K	
MDBS	1.05A		+	48K	
MDBS-DRS	1.02		+	52K	
MDBS-QRS	1.0		+	52K	
MDBS-RTL	1.0		+	52K	
MDBS-PKG			+	52K	w/all above MDBS products
Medical Management System 8000	8.7a		8080		Needs CBASIC
Medical Management System 9000	1.07		8080		Needs CBASIC
Microcosm			Z80		CP/M 2.X or MP/M
Microspell	4.3		8080	48K	Needs 150K/drive
Microspell Demo	**1.0**				**For Dealers Only**
Microstat	2.04		8080	48K	Needs BASIC-80, 5.03 or later
Microstat for Apple	2.0				
Mince	2.6		8080	48K	
Mince Demo	2.6		8080	48K	
Mini-Warehouse Mngmt. Sys.	5.5		8080	48K	Needs CBASIC
Money Maestro		1.1	8080/Z80	48K	CP/M 1.4 or 2.2
MP/M-I	1.0				
MP/M-II	2.0		8080	48K	Needs MP/M
MSORT	1.01		8080	48K	
Mu LISP-80/Mu STAR Compiler	2.10	2.12	8080		
Mu SIMP / Mu MATH Package	2.10		8080		muMATH 80
NAD Mail List System	3.0D		8080	48K	
Nevada COBOL	2.0		8080	32K	
Order Entry w/Inventory/Cybernetics			Z80		Needs RM/COBOL
Panel	2.2			44K	Also for MP/M
PAS-3 Medical	1.77		8080	56K	Needs 132-col. printer & CBASIC
PAS-3 Dental	1.63		8080	56K	Needs 132-col. printer & CBASIC
PASM Assembler	1.02		Z80		
Pascal/M	4.02		8080	56K	
PASCAL/MT Compiler	3.2		8080	32K	
PASCAL/MT+ w/SPP	5.5		8080	52K	Needs 165K/drive
PASCAL/Z Compiler	4.0		8080	56K	
Payroll/Cybernetics, Inc.			Z80		Needs RM/COBOL
Payroll/Peachtree	07-13-81		8080	48K	Needs BASIC-80 4.51
Payroll/Structured Sys	1.0E		8080	60K	w/It Works run time pkg.
PEARL SD	3.01		8080	56K	w/CBASIC2,Ultrasort II
PLAN80 Financial Package (Z80/8080)	2.1a		8080	56K	Z80/8080
PLAN80 Demo	**1.0**				
PL/I-80	1.3		8080	48K	
PLINK I Linking Loader	3.28		Z80	24K	
PLINK-II Linking Loader	1.10A		Z80	48K	
PMATE	3.02		8080	32K	
PRISM/ADS	2.0.1		8080	56K	Needs CBASIC, 2.06 or later & 180K/drive
PRISM/IMS	2.0.1		8080	56K	Needs CBASIC, 2.06 or later & 180K/drive
PRISM/LMS	2.0.1		8080	56K	Needs CBASIC, 2.06 or later & 180K/drive
POSTMASTER Mail List System	**3.5**	**3.5**	**8080**	**48K**	
Professional Time Acctg	3.11a		8080	48K	Needs CBASIC2

(continued next page)

VERSION LIST

Product	S	M	P	MR	
Programmer's Apprentice			8080/Z80	56K	Needs BASIC-80
Property Management Program (AMC)	4.2		Z80	48K	Needs CBASIC 2.07+, CP/M-80 2.0+
Property Management System	07-13-80		8080		Needs BASIC-80 4.51
Property Manager	1.0		8080	48K	Needs CBASIC
PSORT	1.2		8080		
QSORT Sort Program	2.0		8080	48K	
Real Estate Acquisition Programs	2.1		8080	56K	Needs CBASIC
Remote	3.01		Z80		
Residential Prop. Mngemt. Sys.	1.0		Z80	48K	
RM/COBOL Compiler	1.3C		8080	48K	w/Cybernetics CP/M 2
RAID	5.0.2	5.0.2	8080	28K	Modified for TRS-80 Model-I only!
RAID w/FPP	5.0.2	5.0.2	8080	40K	
RECLAIM Disk Verification Program	2.1		8080	16K	
SBASIC	5.4		8080	48K	
Scribble	1.3		8080		
SELECTOR-III-C2 Data Manager	3.24	3.24	8080	48K	Needs CBASIC
SELECTOR-IV	2.17		8080	52K	Needs CBASIC
Shortax	1.2		Z80	48K	TRSDOS,MDOS too, needs BASIC-80 5.0
SID Symbolic Debugger	1.4		8080		N/A-Superbr'n
SMAL/80 Programming System	3.0		8080		For CP/M 1.x
Spellguard	2.0		8080/Z80	32K	Needs Word Processing Program
Standard Tax	1.0		8080	48K	Needs BASIC-80 4.51
STATPAK	1.2	1.2	8080		Needs BASIC-80 4.2 or above
STIFF UPPER LISP	2.6		8080	48K	
STRING BIT FORTRAN Routines	1.02	1.02	8080		
STRING/80 bit FORTRAN Routines	1.22		8080		
STRING/80 bit Source	1.22		8080		
SUPER SORT I Sort Package	1.5		8080		Max. record=4096 bytes
SELECT			8080/Z80	40K	
T/MAKER II	2.4		8080	48K	Avail. for CDOS
T/MAKER II DEMO	2.4		8080	48K	
TEX Text Formatter	2.1		8080	36K	
TEXTWRITER-III	3.6	3.6	8080	32K	
TINY C Interpreter	800102C		8080		
TINY C-II Compiler	800201		8080		
TRS-80 Customization Disk	**1.3C**		**8080**		
ULTRASORT II	4.1B		8080	48K	
Lifeboat Unlock	1.3		8080		Use w/BASIC-80 5.2
VISAM	**2.3p**		**8080**	**48K**	
Wiremaster			Z80		Needs 180K/drive
Wordindex	3.0		8080	48K	Needs WordStar
Wordmaster	1.07A		8080	40K	
WordStar	3.0		8080	48K	
WordStar w/MailMerge	3.0		8080	48K	
WordStar Customization Notes	3.0		8080		
XASM-05 Cross Assembler	1.05		8080	48K	
XASM-09 Cross Assembler	1.07		8080	48K	
XASM-51 Cross Assembler	1.09		8080	48K	
XASM-F8 Cross Assembler	1.04		8080	48K	
XASM-400 Cross Assembler	1.03		8080	48K	
XASM-18 Cross Assembler	1.41		8080		
XASM-48 Cross Assembler	1.62		8080		
XASM-65 Cross Assembler	1.97		8080		
XASM-68 Cross Assembler	2.00		8080		
XYBASIC Extended Interpreter	2.11		8080		
XYBASIC Extended Disk Interpreter	2.11		8080		
XYBASIC Extended Compiler	2.0		8080		
XYBASIC Extended Romable	2.1		8080		
XYBASIC Integer Interpreter	1.7		8080		
XYBASIC Integer Compiler	2.0		8080		
XYBASIC Integer Romable	1.7		8080		
ZAP-80	1.4		8080		Needs 50K/drive
Z80 Development Package	3.5		Z80		N/A-Magnolia,Superbr'n,mod.CP/M
ZDM/ZDMZ Debugger	1.2/2.0		Z80		For N'Star,Apple,IBM 8"
ZDT Z80 Debugger	1.41	1.41	Z80		N/A-Superbr'n,mod.CP/M
ZSID Z80 Debugger	1.4A		Z80		N/A-Superbr'n,mod.CP/M

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

WIN AN IBM* PERSONAL COMPUTER FROM THE NEW MAGAZINE THAT FEATURES THEM.

ANNOUNCING PC™ THE INDEPENDENT GUIDE TO IBM PERSONAL COMPUTERS.

If you're interested in the new IBM Personal Computer, you'll be *very* interested in new *PC* magazine.

Packed with vital information, *PC* will help you keep up with the ever-expanding variety of uses for IBM Personal Computers. *PC* will provide hundreds of helpful tips on adapting Personal Computers for your own needs, and keep you up-to-date on all new developments affecting IBM "PCs" and the people who use them.

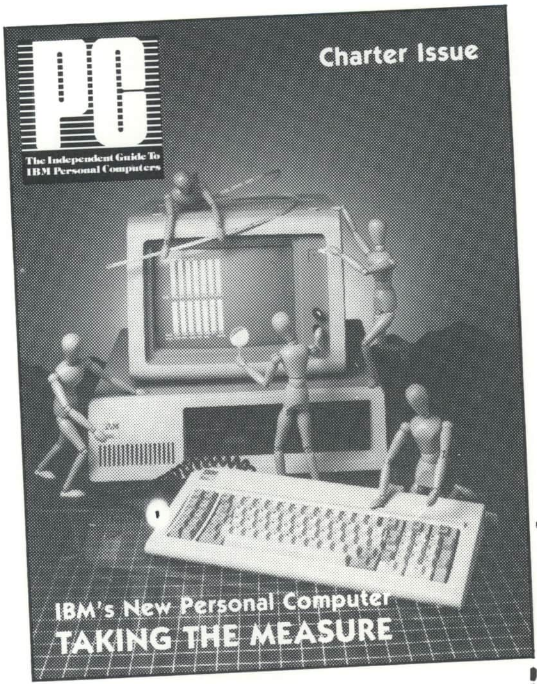
PC even has a special department called PC-Lab™ to evaluate hardware, software and supplies that can be used with IBM Personal Computers—helping you choose those that best suit your needs.

When IBM introduced its Personal Computer, an IBM executive said, "If you could choose one word to describe this system, it would be 'quality.' " That will also be the guiding light for *PC* magazine—in its easy-to-understand writing, its colorful design, and particularly in its devotion to you, the reader.

PC's premiere issue will be out in January, and no one who wants to know more about IBM Personal Computers should be without it—or any of the informative issues to follow. So if you're interested in IBM Personal Computers, be a *PC* Charter Subscriber.

OUR NO-RISK TRIAL OFFER

PC's special offer for Charter Subscribers is the first six issues for just $12, a $6 saving from the single-copy price. And if you're not satisfied with your first issue for any reason, you can


PC
The Independent Guide To IBM Personal Computers
Charter Issue
IBM's New Personal Computer
TAKING THE MEASURE

cancel and get a refund. Simply write "I want a refund" on the mailing label of your first copy, send it back to us within 10 days, and *PC* will refund your payment in full plus 20¢ to pay for your stamp.

ANNOUNCING THE PC "PC" GIVEAWAY.

PC wants to know who's interested in the new IBM "PC," so we're giving one away to help gauge public interest. You could be the one to win our prize—an IBM Personal Computer system unit (16K) with keyboard and color/graphics monitor adapter. (Available IBM products of equal value may be substituted if you prefer.)

You don't have to subscribe to enter our Giveaway drawing; just fill out and mail the "Giveaway" end of the double coupon below. But why not use the other half of the coupon to enter your subscription at the same time. After all, with our no-risk trial offer, you can't lose. And you'll be sure not to miss out on *PC's* information-filled Premiere Issue.

GIVEAWAY RULES

□ Mail entries postpaid to the address on the entry blank. Entries must be postmarked before midnight, March 1, 1982. Drawing will be held, and the winner notified by April 1, 1982. Prize delivery date will be subject to IBM product availabilities. □ Entries must be on an official entry form or its exact copy. □ You may enter as many times as you wish, but each entry must be mailed in a separate envelope. □ Employees of Software Communications, Inc., its affiliates, dealers, distributers, advertising agencies and media not eligible. □ Void where prohibited, taxed or restricted by law.

SUBSCRIBE TO PC NOW, AT NO RISK.

□ **YES I want to be a Charter Subscriber to PC.** My charge information or check for $12 made out to **PC is enclosed.**

PC
1239 21st Avenue
San Francisco, CA 94122

Name ______________________

Address: ______________________

____

City ______________________

State ____________ Zip ____________

□ Visa □ MasterCard □ Check Enclosed

Acct. # ______________________

Exp. Date ______ Bank # (MC only) ______

Signature ______________________

ENTER THE PC "PC" GIVEAWAY.

□ **YES Enter my name in PC's drawing to give away an IBM Personal Computer.**

Area Code/Phone# ______________

I now have: □ an IBM Personal Computer
□ another personal computer
□ no personal computer

*PC is an independent journal, not affiliated in any way with International Business Machines Corporation. IBM is a registered trademark of International Business Machines Corporation. *PC* and PC-Lab are trademarks of Software Communications, Inc.

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

[redacted]