LIFELINES™

The Software Magazine™

$3.00 January 1983 Volume III, No. 8 (ISSN 0279-2575, USPS 597-830)

TIM™ III

The Non-Programming Approach to Data Base Management

Data Base Management

Data management packages were created to save time and money in the development of software solutions to information problems. Many have been designed to accomplish just that, although most have only the programmer in mind. Sure they would save time in the long run, but what of the initial investment in time and effort required to learn the new language? What about the non-programmers in the world who would like an easy yet powerful applications generator? The solution is one of the most highly acclaimed software packages of our time, T.I.M. III.

What is T.I.M.?

T.I.M. is **Total Information Management.** Programmers love it due to its original solutions to classic data management problems. Non-programmers adore it since they can use it to achieve the same results as with other more complicated programming-like packages.

What Makes T.I.M. So Simple to Use?

We at Innovative Software, Inc. designed T.I.M. from day one with the end user in mind. Maybe he is a programmer who doesn't have time to learn a new language. Or perhaps a neophyte who fears coding pads and lines numbered by tens. We felt that a data management package should be able to be used by anyone from a systems analyst to a secretary. That's why T.I.M. takes a full *menu-driven* approach, uses multiple *HELP* screens, and has a manual that sets a new standard in documentation.

The Manual

Many people believe that the manual is just as important as the software itself, a view that we at Innovative Software, Inc. tend to share. The manual for T.I.M. is divided into two sections, the Reference section and the Primer. The Reference section describes all of T.I.M.'s commands and subcommands. This is done in English, not in technical terms or in our own language. Even if you have never seen a computer before in your life, you'll be able to read and understand our manual immediately. The second section is a primer which goes through several examples for you, again in plain English. These true-to-life examples take the beginner by the hand, and instructs him what to do and when. You will be able to see for yourself that T.I.M.'s only limitation is the imagination of the user.

Features of T.I.M.

T.I.M. has all of the features one has come to expect from a data management package, as well as many new ones. For example, a *word processing* interface that allows you to merge information from a T.I.M. file with letters or other documents created by a word processor. Now you can automatically send personalized letters to hundreds or thousands—quickly and easily. T.I.M.'s *Select* command enables you to pull specific information from a file. For example. "All customers who live in a certain ZIP code, whose last name begins with the letter A to L, whose balance due is less than $50.00." A sophisticated *report generator* and even a *list generator* are also included.

How powerful is T.I.M.? With a maximum record size of 2400 characters and the ability to keep up to forty fields sorted properly at all times, T.I.M. is powerful enough to handle just about any application. T.I.M. can handle over 32,000 records per file, and two files can be linked together for reports if your application requires a many-to-one relationship. T.I.M. also includes all of the same editing commands as your word processor, thus making data entry and editing a snap. You can also pull selected records from one file to place them into another. Files may be restructured to add or subtract fields and/or change field lengths or types.T.I.M. even has it's own utility for backing up hard disks onto floppies.

Where to Find T.I.M.

T.I.M. is available from **Lifeboat Associates.** Or you may purchase from us direct by calling 913/383-1089. Either way you will have the finest data management program available.

Available for CP/M,* and IBM PC DOS.**
CP/M version—$695. IBM PC version—$495.

Innovative Software, Inc.
9300 W. 110th Street, Suite 380
Overland Park, Kansas 66210 USA
913/383-1089

TIM is a Trademark of Innovative Software, Inc.
*CP/M and MP/M are Trademarks of Digital Research
**Trademarks of IBM

LIFELINES
The Software Magazine

Editor-in-Chief: Edward H. Currie
Editor: Bonita E. Taylor
Production Manager: Harold Black
Art and Design Manager: Kate Gartner
Marketing & Circulation Manager: Patricia Matthews
Typographer: Harold Black

Editorial Assistant: Patricia Matthews
New Versions Editor: Lee Ramos
Technical Consultants: Al Bloch
Emil Sturniolo
Cover: K. Gartner
Administrative Assistant: Susan Sawyer
Printing Consultant: Sid Robkoff

Opinion

Software Notes

Product Status Reports

Digital Dollars Department

Features

Opinion Editorial

Edward H. Currie

Blessed Be The Unreasonable Among All Men

"The reasonable man adapts himself to the world; the unreasonable one persists in tryng to adapt the world to himself. Therefore all progress depends upon the unreasonable."
–George Bernard Shaw.

The world of microcomputers is heavily populated by what some might deem unreasonable men. These stalwart individuals have steadfastly refused to listen to the so-called reasonable among us.

How else will we explain to our descendants events such as the IBM operating system for the PC having been written by a small company in Seattle, that the first microcomputer was designed by a company determined to bring micros to the masses, that BASIC was in fact an appropriate language for micros, that the floppy was a viable mass storage device for micros, and on, and on, and on...

NCC has been eclipsed by COMDEX as evidenced by the tremendous turnout for the recent show in Las Vegas. Upwards of a thousand booths and some sixty thousand attendees made for the most interesting microcomputer show to date.

Portable machines such as the DOT, Hyperion, Compaq, etc., were everywhere to be seen. Dual processor machines predicated upon Digital Equipment Corporation's approach with Rainbow are definitely on the increase. It is reasonable that unreasonable men will provide future designs with multiple processors to permit the end user to implement the widest possible range of software.

Packaging continues to be a major area of innovation for both hardware and software as evidenced by the fact that millions of dollars are being expended to follow IBM's lead. Both hardware and software are being given particular attention in the area of cosmetics.

Additional media formats are emerging rapidly as Sony and others offer three-inch floppys.Low profile drives proliferate and there seems to be no end in sight to the increased capacity of Winchesters and their floppy counterparts.

Printer prices are plummeting as more features, including color, are added. Better graphics features are now accessible and are rapidly being exploited. Cipher has introduced a 3M cartridge tape subsystem. This intriguing new product is designed to be plug compatible with five-and-a-quarter-inch drives. It appears to the micro as a slow floppy with either twenty or forty megabytes of storage and it's available in OEM quantities for as little as three hundred dollars! The subsystem accepts the track and sector needed just as a floppy does. Equivalent track to track time is thirty milliseconds. This would be ideal for data bases which are sure to soon be available for micros in a variety of types and media formats. The Z8000 has been given at least a brief reprieve with the intro of Olivetti's Z8000 machine. Faster Z80's continue to appear with the latest entry at 10 Megahertz. If only the peripherals could keep up with these increased processor speeds. There is little to be gained if the peripherals don't offer increased data transfer rates.

The 68000 is beginning to be a reality and is being followed rapidly by applications software to be run under UNIX. Currently, it looks as if UNIX will be the choice for the 68000. This could mean that the rapid diffusion of 68000 based machines will be slowed due to an insufficient availability of application programs which are UNIX compatible.

Digital Research is offering a version of CP/M called CP/M-68K with the hope that C application programs will be moved into the CP/M environment.

Modems are now appearing in a variety of formats and sophisticated telecommunications packages will make it easier for end users to realize the full potential of remote access. Microsoft has introduced a nifty flight simulator which gives you the eerie feeling that you are in fact air borne. Voice recognition and synthesis continues its inexorable march towards perfection and this progress is made manifest in a number of new products. Peachtree has introduced a board which will be integrated with their applications packages, which has an uncanny ability to simulate the human voice.

Softcards are on the increase as Digital Research introduces their Z-80 softcard with CP/M-3 (now known as CP/M-Plus). PCPI has acquired the rights to Coprocessor's 8088 card for the Apple. Microsoft is rumored to have some surprises of their own in the hardware area.

Sadly, no one has yet attempted to capitalize on the design of the Grid by designing a standalone machine of similar configuration. This seems such an obvious approach that it's difficult to understand why no one has adopted it, but perhaps at NCC...

Bubble memory is beginning to appear for machines such as the Apple. Additional Apple lookalikes are also appearing. A large number of plug-in boards are also available for the IBM PC, many of which are reminiscent of similar S100 boards.

Telex replacements in both hardware and software were also prevalent at COMDEX. This is an obvious use for micros and hopefully will spell the end of the Teletype dinosaurs. Like nine track magnetic tape - or even worse, paper tape - old computer stuff never dies, it just slowly fades away.

Books on computers seem to be appearing with the frequency of new application programs. It's interesting to note that books on BASIC are still the number one seller. *Future Computing* claims that an aspiring author can make a very healthy income by

(continued on page 36)

Feature

Sliding into BDOS, Part II With Files Made Easy

Michael J. Karas

In this second part of our tutorial on using files with the CP/M BDOS, I will not reiterate the importance of the CP/M BDOS file interface. Nor will I try to explain the tutorial's value. If you are new on the scene and have some questions, I would like to direct your attention to the November 1982 issue, where the first part of this series was presented. There the purpose of the BDOS and the general interface concepts were presented. The article went on to include a description of the physical device system calls and other miscellaneous system control type functions.

This Time: Files

This month the tutorial continues by describing the sequential file I/O system supported within the BDOS. The concepts of CP/M file storage will also be covered, along with appropriate CP/M directory structure definition as it relates to the file access on a CP/M disk. The FILE CONTROL BLOCK (FCB) will be described in terms of its functions as related to disk file access. I have also included a comprehensive programming example which allows a sequential file to be accessed character by character.

How Files Are Stored On Disk

The CP/M operating system manages the available space on a disk by dividing the total available space into a number of relatively small data block storage areas called "GROUPS". A group size is usually described as the minimum allocatable space that a file can occupy. This means that the operating system lumps sets of the normal 128 byte logical records of a file together into these groups. The number of groups contained on a disk derives from the total file storage space of the disk in logical 128 byte records divided by the number of 128 byte logical records lumped together into a group. (A note to the less casual reader: the number of groups on a disk is limited by design to 64K groups. Secondly, a group is always an integral power-of-two number of 128 byte logical records with a minimum size of 8 records [1K byte], Group size is necessarily limited to 16K bytes, due to the extent system described below.)

As a file is stored on a CP/M disk it consumes disk space in 128 byte logical records. Each time a group becomes filled with records the operating system allocates another group to the file. Hence the term "minimum allocatable size." If, as the file grows in size, the last allocated group assigned to a file is not completely filled, the remaining space in the group is "burned in" – not usable by other files. Through a stored directory, the CP/M system keeps track of the group assignments made to the files on a disk, the file names, and the total number of 128 byte logical records in each file. The first portion of the disk is reserved for the file directory. A fixed number of directory entries, determined by the system's BIOS design, are available, usually a number like 64, 128, or 256, depending upon disk size.

Each file has a unique directory entry "set" describing the location on the disk. A "set" of directory entries is specified because each entry is designed to "point to" or store the group allocation numbers for that file. Each directory entry has a number slot where group numbers can be stored and each entry can specify the storage for 16K bytes of space. For files larger than 16K bytes a separate directory entry is used for each 16K bytes (or remainder thereof), and each such piece of a file is called an "EXTENT." The directory entry "set" for a file contains a byte in each extent directory entry that stores the extent number of the file. Extent numbers start with 0 and may increase to a theoretical limit of 255 or the size of the disk in 16K byte pieces, whichever is smaller.

The chart in Figure 1 describes the functions of all bytes in a typical directory entry. Each entry is 32 bytes long and they are packed four to a logical sector; the number of logical sectors filled up with directory entries is limited to the predetermined number of directory entries divided by four.

Figure 1. DISK DIRECTORY ENTRY DEFINITION

byte 00	byte 01	byte 02	byte 03	byte 04	byte 05	byte 06	byte 07
Active Entry & User Flag	Eight Character ASCII File Name Bytes 01 to 08						

byte 08	byte 09	byte 10	byte 11	byte 12	byte 13	byte 14	byte 15
Last File Name Char	Three character ASCII File Name extension			Extent Number	Two Bytes Reserved		Record Count of this Extent

byte 16	byte 17	byte 18	byte 19	byte 20	byte 21	byte 22	byte 23
Group Number Storage for groups attached to this file One byte used per group number if disk contains less than 255 groups. Two bytes if greater than 256.							

byte 24	byte 25	byte 26	byte 27	byte 28	byte 29	byte 30	byte 31
Additional Group Number storage. Group Number storage for groups attached to this file One byte used per group number if disk contains less than 255 groups. Two bytes if greater than 256.							

The bytes of the disk directory entry are described in the following paragraphs. The first byte stored in an entry is set to indicate whether this slot in the predetermined directory area is empty or if it describes an active file extent. A value of 0E5H indicates an empty slot. Presumably, this value was selected because a freshly formatted diskette contains all 0E5H bytes in the empty sectors, making it appear to contain no files. If the byte value is not 0E5H, then the slot contains a valid file extent descriptor. The CP/M user number area associated with an active file is stored in the first directory entry byte. User number values range from 0 to 15.

The next eight bytes contain the primary name of the file in ASCII characters. If the name is shorter than eight characters, it is padded to the right with spaces. Following the name field is a three byte file name extension field in ASCII characters. The extension field, if shorter than three characters, is padded to the right with spaces. In CP/M version 2.2, the upper bits (bit 7) of the extent name bytes are used to describe certain file attributes. If the upper bit of the first extent name character is set, then the file is described as a read-only file. The upper bit of the second extent name character, if set, indicates that the file name should not be displayed in directory listings.

The byte next after each directory entry (as a file descriptor extent) is set to a number specifying which 16K byte chunk the entry describes. Two bytes after the extent byte are not used within the directory and are normally set to zero by default. The number of records stored in the extent this directory entry describes is recorded in the byte 15 position. The maximum value for the record count is 128 (080H) which is equal to (128 * 128) or 16K bytes, the maximum size of an extent.

Byte positions 16 to 31 contain the disk group numbers which contain the data belonging to the file named in the directory entry. The number of bytes used for group number storage within the total 16 available is dependent upon the amount of file data described by this extent and by the group size of the disk. The group numbers are single byte numbers, up to 16 total, if the number of groups on the disk is less than or equal to 255. If the number of groups on the disk is more than 255, byte positions 16 to 31 contain two byte group numbers, stored in low byte/high byte order. The group numbers contained within a directory entry do not have to be in increasing sequential order nor do they have to be consecutive.

Figure 2 shows two logical records of the directory from a single-sided double density disk with 2K byte groups. The total number of groups available is 243, so the group numbers are single byte numbers. Note that only one half the 16 byte space for group numbers is used, because for 2K byte groups only eight entries are needed to describe the storage for one full 16K byte extent.

Figure 2. EXAMPLE HEX/ASCII DIRECTORY RECORD DISPLAY

```
00   00414449 52202020  20434F4D 0000000B   .ADIR    COM....
10   07000000 00000000  00000000 00000000   ................
20   004D4552 47505249  4E4F5652 0000003C   .MERGPRINOVR...<
30   16171819 00000000  00000000 00000000   ................
40   00434F50 59202020  20434F4D 0000000E   .COPY    COM....
50   0C000000 00000000  00000000 00000000   ................
60   00435243 4B202020  20434F4D 0000000A   .CRCK    COM....
70   0D000000 00000000  00000000 00000000   ................

00   E5555345 52202020  204C4F47 00000030   eUSER    LOG...0
10   04050600 00000000  00000000 00000000   ................
20   00444454 20202020  20434F4D 00000026   .DDT     COM...&
30   0F101100 00000000  00000000 00000000   ................
40   0044552D 56373520  20434F4D 0000002E   .DU-V75  COM....
50   12131400 00000000  00000000 00000000   ................
60   00464F52 4D415420  20434F4D 0000000C   .FORMAT  COM....
70   15000000 00000000  00000000 00000000   ................
```

The Figure 2 examples all show files that are fewer than 16K bytes each. Note also the display showing the erased "USER.LOG" file.

How Files Are Accessed

Disk files are accessed through a user description block called a File Control Block (FCB for short). The file control block, used by virtually all file access BDOS system calls, has the structure as shown in Figure 3 (see page 16). This chart is taken from a Digital Research CP/M manual and is included here for quick educational reference.

Note that the structure of a file control block is much the same as that of a directory entry with a few minor changes. The changes and/or differences are as follows, otherwise the byte descriptions are the same as for the disk directory entry.

The first byte of an FCB allows the programmer to specify which drive should be used for the file access. Drive A; to P: are specified as 1 to 16 respectively, while a value of zero indicates that the currently logged default drive should be used for the access.

An FCB contains four additional bytes that are used as pointers for file access position. The "cr," current record number, indicates the sequential record number of this extent that will be accessed upon the next file read or file write system call. The user normally sets the "cr" byte to zero to begin file access at the first logical record of the file. Each time a read or write is performed the current record number is incremented. When the "cr" byte attains a value of 080H during a sequential file operation the BDOS automatically realizes that the current extent of the file has been fully accessed and performs the necessary disk directory accesses to setup the FCB to allow file access to the next extent. For reading this simply means that the next extent descriptor directory entry from the disk, for this file, is read into memory (i.e., the group allocation numbers from the disk are copied into the d0-dn bytes of the FCB, the extent number becomes one greater, the record count from the disk for the new extent is copied into the "rc" byte and the cr byte is zeroed) During a writing operation the "cr" byte attaining a value of 080H indicates that the current extent of the file is full and so the BDOS automatically finds the appropriate directory entry spot on the disk to write in the newly assigned group allocation bytes, record count value and extent number. The BDOS will then create another directory entry on the disk for the new extent of the file. In this case the d0-dn bytes of the FCB are zeroed to indicate that storage has not yet been allocated for this extent.

The last three bytes of the FCB, r0, r1, and r2 are for random record file I/O and will be covered in the third part of this tutorial. For simpler sequential I/O the FCB does not even need to be set up for the 36 bytes of storage. 33 bytes suffice for all sequential file I/O FCB operations.

File Access Set Up System Functions

The procedure for accessing a file generally starts in one of two ways. The first scenario starts with, "Let's see if our file exists on the disk?" There are two BDOS system calls

(continued on next page)

related to the functions of searching the disk directory for a file name match against the FCB specified by the user. These operations allow for the programmer to find out whether a specific file name already exists upon the disk. In addition, they provide a mechanism for scanning a directory to determine all file names that exist in it.

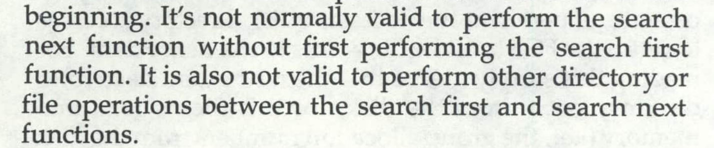
The second situation arises if the programmer is already aware of the file status with respect to "presence" on the disk. These latter functions are used to work with specific files for opening, closing, creating, renaming and deleting.

SEARCH FIRST AND SEARCH NEXT: Functions 17 and 18.

The search functions scan the directory to match a file name comparing with the user-specified FCB pointed to by the (DE) register pair. The match is made on the basis of comparing the f1-f8, t1-t3, and ex bytes of the FCB to the corresponding bytes of the disk directory entries. Any FCB position that contains an ASCII question mark "?" (03fH) is specified as a "match any character" from the disk directory. The function calls return a value of 0FFH in the (A) register if no more matched directory entries can be found. The search functions cause the currently valid disk buffer address and the following 128 bytes to be filled with a copy of the directory record containing the matched entry, if one is found. The (A) register is returned with a 0 to 3 value to indicate which one of four possible 32 byte chunks of the directory record contain the matched entry.

Search first finds the first occurence of a matched entry to the FCB. The search next function scans the directory from the current search position instead of from the beginning. It's not normally valid to perform the search next function without first performing the search first function. It is also not valid to perform other directory or file operations between the search first and search next functions.

The program example below shows a technique for reading all directory entries from the disk drive specified by the first FCB byte into a memory resident list. The list starts at the LIST label with the total matched file resident list. The list starts at the LIST label with the total matched file count stored in the FILECNT variable. The LISTPOS label stores the next available list load point during the directory scan operation. The search FCB uses the CP/M default FCB location at address 05CH and specifies a total wild card (*.*) match. The "ex" byte is zeroed before the search first call so that only the zero extents of the files are returned. The file names are stored in the list in character strings of 16 bytes each, with a preceding drive designator byte and padded to the right with four zero bytes. Please note that this program is a segment only and will not directly assemble and run as a CP/M .COM file without lead in and error exit coding.



Listing 1. A DIRECTORY SCANNING PROGRAM

```
BUFR    EQU     80H+BASE ;DEFAULT CP/M BUFFER
BDOS    EQU     0005H    ;ENTRY POINT FOR
                         ;BDOS OPERATIONS
SRCHF   EQU     17       ;SEARCH DIR FOR 1ST OCCUR.
SRCHN   EQU     18       ;SEARCH DIR FOR NXT OCCUR.
STDMA   EQU     26       ;SET DMA ADDRESS
;
FCB     EQU     5CH+BASE ;DEFAULT FILE CNTRL BLOCK
FCBEXT  EQU     FCB+12   ;EXTENT BYTE IN FCB
FCBRNO  EQU     FCB+32   ;RECORD NUMBER IN FCB
;
;
;SETUP SIZE OF ELEMENTS IN THE FILE NAME LIST
;
ITEMSZ  EQU     16       ;EA LIST ITEM IS 16 BYTES
;
;
;SETUP WILD CARD FILE IMAGE LIKE *.*
;
        LXI     H,FCB+1  ;LOC TO PUT WLD CRD IMAGE
        MVI     B,11     ;SIZE TO SET
ALFN:
        MVI     M,´?´    ;PUT IN JOKER CHAR
        INX     H        ;BUMP FILL POINTER
        DCR     B        ;DCR BYTE COUNTER
        JNZ     ALFN
;
;
;ZERO INITIAL TOTAL FILE COUNT
;
        LXI     H,0000H
        SHLD    FILECNT
;
;
;HERE IF NAME PROPERLY POSITIONED IN
;THE DEFAULT FCB AREA FOR LIST BUILD
;
NAMEPRES:
        MVI     C,STDMA  ;INITIALIZE DMA ADDRESS
        LXI     D,BUFR   ;TO DEFAULT BUFFER
        CALL    BDOS
;
        XRA     A        ;CLEAR APPROPRIATE FIELDS
        STA     FCBEXT   ;OF SEARCH FCB EXTENT
        STA     FCBRNO   ;BYTE AND RECORD NUMBER
;
        LXI     D,FCB    ;USE DFLT FCB FOR SEARCH
        MVI     C,SRCHF  ;SEARCH FOR 1ST OCCURRENCE
        CALL    BDOS
        CPI     0FFH     ;SEE IF FOUND
        JNZ     LOADLIST ;IF SOME FOUND THEN GO
                         ;BUILD LIST
;
;PUT INSTRUCTIONS HERE TO HANDLE SITUATION WHERE NO
;FILES MATCHING FCB WILD CARD IMAGE ARE FOUND.
;
        JMP     ERROR$EXIT;TO USER SUPPLIED ROUTINE
;
;
;BUILD UP LIST WITH ALL FOUND ENTRIES
;
LOADLIST:
        LXI     H,LIST   ;INIT LIST POINTER PARAMETERS
        SHLD    LISTPOS  ;START = CURRENT POS OF LIST
;
;
;PUT CURRENTLY FOUND NAME TO LIST
;(A) = OFFSET IN DEFAULT BUFFER OF NAME
;
;
NM2LST:
        ANI     3        ;ZERO BASED TWO BIT INDEX
        ADD     A        ;TIMES 32 TO MAKE POSITION
        ADD     A        ;INDEX
        ADD     A
        ADD     A
        ADD     A
        MOV     C,A      ;PUT IN BC
        XRA     B        ;CLEAR HIGH ORDER
        LXI     H,BUFR   ;TO NAME PSTN IN DFLT BUFFER
        DAD     B        ;(HL) = CURRENT FOUND NAME PNTR
        LDA     FCB      ;PUT DISK DRIVE # IN NAME PLACE
        MOV     M,A      ;INTO BUFFER
        XCHG
        LHLD    LISTPOS ;POINTER TO CURRENT
        XCHG            ;LOAD POINT IN LIST
        MVI     B,12    ;MOVE DR DESIGNATOR
MOVLP:                  ;AND NAME TO LIST
        MOV     A,M     ;GET NAME BYTE FR/DEFAULT BUFFR
        STAX    D       ;PLACE INTO LIST
        INX     H       ;BUMP POINTERS
        INX     D
        DCR     B       ;CHECK MOVE BYTE COUNT
        JNZ     MOVLP
        XCHG            ;(DE) WAS LEFT WITH NEXT
                        ;LOAD POINT ADDRESS
        MVI     B,ITEMSZ-12 ;REST OF LIST ITEM
FILZRO:                 ;SPACES TO ZERO OUT
        MVI     M,00H   ;PUT IN A ZERO BYTE
        INX     H
        DCR     B       ;ALL REST FILLED YET
        JNZ     FILZRO
;
        SHLD    LISTPOS ;KEEP NXT LOAD PNT IN SAFE LOC
        LHLD    FILECNT ;INCREASE COUNT FOR EA FILE
        INX     H
        SHLD    FILECNT
;
;SEARCH FOR NEXT OCCURRENCE OF SPECIFIED FILE NAME
;
        MVI     C,SRCHN ;SEARCH NEXT FUNCTION CODE
        LXI     D,FCB   ;FILE NAME SPECIFICATION FIELD
        CALL    BDOS
        CPI     0FFH    ;SEE IF ALL THRu DIRECTORY YET
        JNZ     NM2LST  ;IF NOT GO PUT NAME INTO LIST
;
;
;PROGRAM EXECUTION TO HERE IF THE LIST CONTAINS SOME
;FILE NAMES FROM THE DISKETTE
;
;USER DOES OWN THING FROM HERE
;
;
;DIRECTORY NAME LIST FOR STORAGE OF INPUT NAMES
;
FILECNT:
        DS      2       ;COUNTER FOR NUMBER OF FILES

LISTPOS:
        DS      2       ;STORAGE FOR CURRENT LIST
                        ;LOAD POINTER
;
LIST:
        DS      1       ;START POINT FOR FILENAME LIST
;
;+++...END OF LISTING 1.
```

OPEN FILE: Function 15.

An existing disk file may not be read until the user FCB contains information about where the file is stored. Function 15 provides a means for the user to fill in the file name and then calls the operating system to get the d1-dn bytes of the FCB filled in. Once the file is OPEN, it may be read because subsequent calls to the BDOS to READ will "know where" the file is located. The OPEN function returns a value of 0FFH if the file cannot be found, otherwise the (A) register contains a value of 0 to 3 to indicate that the file was successfully opened. To open a file the programming procedure is simply:

```
;
;OPEN FILE EXAMPLE
;
OPEN        EQU     15    ;OPEN FUNCTION CODE
BDOS        EQU     0005H ;SYSTEM ENTRY

            ORG     0100H ;START
            LXI     D,FCB ;POINT AT FCB
            MVI     C,OPEN ;FUNCTION
            CALL    BDOS
            CPI     0FFH  ;CHECK IF NOT FOUND
            JZ      ERROR
            RET           ;IF OPEN GO TO CCP
;
ERROR:
            MVI     C,9   ;PRINT ERROR MESSAGE
            LXI     D,ERRMS
            CALL    BDOS
            RET
;
ERRMS:
            DB      ´FILE NOT FOUND´,´$´
;
;
;FILE ACCESS FILE CONTROL BLOCK
;
FCB:
            DB      00H   ;SET TO USE DEFAULT DRIVE
            DB      ´TEST    DAT´,0,0,0,0
            DS      16    ;STORAGE FOR D1 TO DN BYTES
            DB      0     ;CURRENT RECORD BYTE
;
            END
```

CLOSE FILE: Function 16.

When a file is accessed for writing, new space is allocated for that file on the disk. This implies that the user FCB contains disk group numbers that are not stored upon the diskette in the directory entry for the file. Function 16 provides a means for the user to complete the file writing operation and then call the operating system to set the directory entry group allocation bytes, the rc byte and the extent byte from the corresponding bytes of the FCB. A file that has been opened for reading only need not be closed, because there is no change in the stored disk directory information. The CLOSE function returns a value of 0FFH if the file cannot be found; otherwise the (A) register contains a value of 0 to 3 to indicate that the file was successfully closed. To close a file the programming procedure is simply:

```
;
;CLOSE FILE EXAMPLE
;
CLOSE       EQU     16    ;CLOSE FUNCTION CODE
BDOS        EQU     0005H ;SYSTEM ENTRY

            ORG     0100H ;START
            LXI     D,FCB ;POINT AT
            MVI     C,CLOSE   ;FCB FUNCTION
            CALL    BDOS
            CPI     0FFH  ;CHECK IF NOT FOUND
            JZ      ERROR
            RET           ;IF CLOSED GO TO CCP
;
ERROR:
            MVI     C,9   ;PRINT ERROR MESSAGE
            LXI     D,ERRMS
            CALL    BDOS
            RET

;
ERRMS:
            DB      ´FILE NOT FOUND´,´$´
;
;FILE ACCESS FILE CONTROL BLOCK
;
FCB:
            DB      00H   ;SET TO USE DEFAULT DRIVE
            DB      ´TEST    DAT´,0,0,0,0
            DS      16    ;STORAGE FOR D1 TO DN BYTES
            DB      0     ;CURRENT RECORD BYTE
;
            END
```

DELETE FILE: Function 19.

Often the programmer will create and write files which will subsequently not be needed. The file or files may be deleted through use of function 19. The user sets an FCB to the appropriate file name in the f1-f8 and t1-t3 bytes. The

(continued on next page)

BDOS function then removes the specified file from the directory. The user specified file name in the FCB may contain ASCII question marks, in which case the delete function may delete multiple files if the file name matches more than one file on the disk. The "?" matches any character at the position of its occurrence in the name. The DELETE function returns a value of 0FFH if the file(s) cannot be found, otherwise the (A) register contains a value of 0 to 3 to indicate that the file was successfully deleted. To delete a file the programming procedure is simply:

```
;
;DELETE FILE EXAMPLE
;
DELETE    EQU       19    ;CLOSE FUNCTION CODE
BDOS      EQU       0005H ;SYSTEM ENTRY

          ORG       0100H ;START
          LXI       D,FCB ;POINT AT FCB
          MVI       C,DELETE  ;FUNCTION
          CALL      BDOS
          CPI       0FFH  ;CHECK IF NOT FOUND
          JZ        ERROR
          RET             ;IF CLOSED GO TO CCP
;
ERROR:
          MVI       C,9   ;PRINT ERROR MESSAGE
          LXI       D,ERRMS
          CALL      BDOS
          RET
;
ERRMS:
          DB        ´FILE NOT FOUND´,´$´
;
;FILE ACCESS FILE CONTROL BLOCK
;
FCB:
          DB        00H   ;SET TO USE DEFAULT DRIVE
          DB        ´TEST    DAT´,0,0,0,0
          DS        16    ;STORAGE FOR D1 TO DN BYTES
          DB        0     ;CURRENT RECORD BYTE
;
          END
```

CREATE FILE: Function 22.

Whenever a new file is desired it must be created so that there is a spot in the directory to later save the file allocation information (see close function above). The BDOS assumes that the programmer has specified a file name that does not exist upon the disk. If there is a chance that a new file name may duplicate a name already on the disk, the previously described delete function should be used to erase the old file before creating the new one. Otherwise the directory may contain two files by the same name.

The CREATE function returns a value of 0FFH if there is no room in the directory to store the freshly created directory entry; otherwise the (A) register contains a value of 0 to 3 to indicate that the file was successfully created. A newly created file may be immediately written since the BDOS prepares the user FCB to look like an empty file. To create a file the programming procedure is simply:

```
;
;CREATE FILE EXAMPLE
;
CREATE    EQU       22    ;CREATE FUNCTION CODE
BDOS      EQU       0005H ;SYSTEM ENTRY

          ORG       0100H ;START
          LXI       D,FCB ;POINT AT FCB
          MVI       C,CREATE  ;FUNCTION
          CALL      BDOS
          CPI       0FFH  ;CHECK IF
          JZ        ERROR ;DIRECTORY FULL
          RET             ;IF CLOSED GO TO CCP
;
ERROR:
          MVI       C,9   ;PRINT ERROR MESSAGE
          LXI       D,ERRMS
          CALL      BDOS
          RET
;
ERRMS:
          DB        ´DIRECTORY FULL´,´$´
;
;
;FILE ACCESS FILE CONTROL BLOCK
;
FCB:
          DB        00H   ;SET TO USE DEFAULT DRIVE
          DB        ´TEST    DAT´,0,0,0,0
          DS        16    ;STORAGE FOR D1 TO DN BYTES
          DB        0     ;CURRENT RECORD BYTE
;
          END
```

RENAME FILE: Function 23.

Sometimes it is necessary to change the name of a disk file from that in the disk directory. With function 23 the user specifies the name of an existing file on the disk with a standard FCB format, but on calling the BDOS the d1-dn byte area of the FCB is set to the new name. All occurences of the existing file name (i.e., all extents) are changed to match the new name. The drive select byte specifies the drive on which the rename operation should be performed. The first byte of the second 16 bytes of the FCB (d0) is expected to be zero. The RENAME function returns a value of 0FFH if the old name file could not be found, otherwise the (A) register contains a value of 0 to 3 to indicate that the file was successfully renamed. To rename a file the programming procedure is simply:

```
;
;RENAME FILE EXAMPLE
;
RENAME    EQU       23    ;RENAME FUNCTION CODE
BDOS      EQU       0005H ;SYSTEM ENTRY

          ORG       0100H ;START
          LXI       D,FCB ;POINT AT FCB
          MVI       C,RENAME  ;FUNCTION
          CALL      BDOS
          CPI       0FFH  ;CHECK IF
          JZ        ERROR ;DIRECTORY FULL
          RET             ;IF CLOSED GO TO CCP
;
ERROR:
          MVI       C,9   ;PRINT ERROR MESSAGE
          LXI       D,ERRMS
          CALL      BDOS
          RET
;
ERRMS:
          DB        ´FILE NOT FOUND´,´$´
;
;FILE ACCESS FILE CONTROL BLOCK
;
FCB:
          DB        00H   ;SET TO USE DEFAULT DRIVE
          DB        ´TEST    DAT´,0,0,0,0 ;OLD NAME
          DB        00H   ;BYTE ASSUMED TO BE ZERO
          DB        ´NEWNAME DAT´,0,0,0,0 ;NEW NAME
          DB        0     ;CURRENT RECORD BYTE
;
          END
```

Accessing File Data

The previous section showed the reader how to find and set up files for subsequent I/O. Other file/directory handling functions were also presented, leading up to the big moment when the program is finally ready to read or write data from/to a disk file. So here it is at last. . .

CP/M disk file data is moved between the disk and memory in blocks of 128 bytes called logical records, or "sectors" in older CP/M lingo. Two functions presented here are included in the CP/M BDOS function code to allow sequential access to blocks of data in a file. The READ function starts at the beginning of a file and reads data blocks to the end of the file. The opposing WRITE operation moves data blocks to a new disk file and writes to the end of the user's data when the file is closed (or the disk is full if the programmer has too much data). The BDOS includes one other function that allows the user to specify the area in the program where the 128 byte disk record buffer is to be located. These three functions will be described below.

SET DISK BUFFER ADDRESS: Function 26.

The 128 byte data buffer used by the BDOS for file I/O is based at an address commonly called the "DMA ADDRESS." This address or "buffer pointer" is passed to the BDOS in the (DE) registers when performing function 26. The program below simply sets the buffer address to "DATBF", a storage area after the end of the short program.

```
;
;SET BUFFER ADDRESS EXAMPLE
;
STDMA      EQU       26    ;SET BUFFR ADDR FUNCTION CODE
BDOS       EQU       0005H ;SYSTEM ENTRY

           ORG       0100H ;START
           LXI       D,DATBF ;POINT AT DATA BUFFER
           MVI       C,STDMA ;FUNCTION
           CALL      BDOS
           RET                ;BACK TO CCP
;
DATBF:
           DS        128    ;SETUP 128 BYTE BUFFER
;
           END
```

READ AND WRITE DISK RECORDS: Functions 20 and 21.

The disk read and write functions are very similar in operation in that both move 128 bytes of data to/from the user's program. The READ assumes entry with (DE) pointing to an active FCB setup by the open file function. The read sequential function reads the 128 byte record specified by the "cr" field of the FCB into the buffer pointed to by the current disk buffer address. After each READ operation the "cr" field is incremented to the next record number. If the "cr" field overflows past the end of the extent without encountering the end of the file, then the BDOS automatically opens the next extent in preparation for the next read operation. The READ function returns a 00H code in the (A) register if the READ was performed successfully. If the end of file is encountered, a non-zero value is returned in (A).

The WRITE function assumes, on entry to the BDOS, that the (DE) registers point at a validly opened or created FCB. The WRITE will move 128 bytes of data from the buffer specified by the current disk buffer address to the disk. The written record is placed at the "cr" record position of the extent. As each record is written, the "cr" field is incremented in preparation for the next write operation. As in the READ function, if the "cr" field overflows past the end of the current extent, the BDOS automatically closes the current extent and creates a new extent in preparation for the next write operation. The WRITE command may be performed on an existing file. If the file currently contains data at the "cr" record, the WRITE will overlay the current data with the new 128 byte record. The WRITE function returns a 00H value in the (A) register if the operation is successful; a non-zero value is returned if the write function was unsuccessful due to a full disk or directory.

The small program below is designed to read the first record of a file 'TEST.DAT', and write it into the small file 'ONEREC.DAT'. The program should be self documenting.

```
;
;READ AND WRITE FUNCTION EXAMPLES
;
READ       EQU       20    ;READ FUNCTION CODE
WRITE      EQU       21    ;WRITE FUNCTION CODE
OPEN       EQU       15    ;OPEN FUNCTION CODE
CLOSE      EQU       16    ;CLOSE FUNCTION CODE
DELETE     EQU       19    ;DELETE FUNCTION CODE
CREATE     EQU       22    ;CREATE NEW FILE
STDMA      EQU       26    ;SET DISK BUFFER ADDRESS
BDOS       EQU       0005H ;SYSTEM ENTRY

           ORG       0100H ;START
           LXI       D,DATBF ;POINT AT DATA BUFFER
           MVI       C,STDMA ;FUNCTION
           CALL      BDOS
;
           LXI       D,FCBIN ;POINT AT AND
           MVI       C,OPEN  ;OPEN INPUT FILE
           CALL      BDOS
           CPI       0FFH    ;CHECK FOR OPEN ERROR
           JZ        ERROR
;
           LXI       D,FCBOUT ;DEFAULT DEL OF NEW FILE
           MVI       C,DELETE ;IN CASE IT
           CALL      BDOS     ;EXISTS ALREADY
           LXI       D,FCBOUT ;POINT AT FCB
           MVI       C,CREATE ;FUNCTION TO MAKE
           CALL      BDOS     ;NEW FILE
           CPI       0FFH     ;CHECK IF DIR FULL
           JZ        ERROR
           XRA       A      ;CLEAR INPUT CR FIELD TO
           STA       INCR  ;READ FIRST RECORD
           LXI       D,FCBIN ;READ FIRST FILE
           MVI       C,READ
           CALL      BDOS
           ORA       A      ;CHECK IF READ WAS O.K.
           JNZ       ERROR
           LXI       D,FCBOUT ;WRITE TO OUTPUT FILE
           MVI       C,WRITE
           CALL      BDOS
           ORA       A      ;CHECK THAT DISK WASNT FULL
           JNZ       ERROR
;
           LXI       D,FCBOUT ;CLOSE OUTPUT FILE
           MVI       C,CLOSE
           CALL      BDOS
           CPI       0FFH   ;CHECK CLOSE STATUS
           RNZ              ;BACK TO CCP IF NO ERROR
;
ERROR:
           MVI       C,9    ;PRINT ERROR MESSAGE
           LXI       D,ERRMS
           CALL      BDOS
           RET
;
ERRMS:
           DB            ´PROGRAM FILE ERROR´,´$´
;
;
;FILE ACCESS FILE CONTROL BLOCKS
;
```

(continued on next page)

```
FCBIN:
        DB      00H  ;SET TO USE DEFAULT DRIVE
        DB      ´TEST    DAT´,0,0,0,0
        DS      16   ;STORAGE FOR D1 TO DN BYTES
INCR:
        DB      0    ;CURRENT RECORD BYTE
;
FCBOUT:
        DB      00H  ;SET TO USE DEFAULT DRIVE
        DB      ´ONEREC  DAT´,0,0,0,0
        DS      16   ;STORAGE FOR D1 TO DN BYTES
        DB      0    ;CURRENT RECORD BYTE
;
DATBF:
        DS      128  ;SETUP 128 BYTE BUFFER
;
        END
```

Sequential File I/O Programming Example

The assembly language code of Listing 2 presents a comprehensive set of I/O routines that allow either an input or output sequential file to be processed on a byte by byte basis. The routines perform all necessary sector buffering. The reader is encouraged to fully study the code and gain an understanding of how it all works. The program uses most of the BDOS functions presented in this tutorial.

```
;

     Listing 2. CHARACTER BY CHARACTER DISK I/O ROUTINES

;
;DEMONSTRATION SEQUENTIAL CP/M FILE
;CHARACTER BY CHARACTER I/O ROUTINES.
;NOTE THAT THE MAIN BODY
;OF THIS PROGRAM IS NOT DESIGNED TO
;RUN AS IS IN ANY NORMAL MANNER.
;
;MANY THANKS DUE TO WARD CHRISTENSEN
;WHO PREPARED THE ORIGINAL SET OF
;SIMILAR I/O ROUTINES BURIED INSIDE
;THE CP/M USERS GROUP MODEM PROGRAM
;THAT HAS BECOME SO VERY POPULAR.
;THANKS AGAIN WARD.
;
;CP/M BDOS EQUATES
;
RDCON   EQU     1
WRCON   EQU     2
PRINT   EQU     9
OPEN    EQU     15              ;OPEN FILE
CLOSE   EQU     16              ;CLOSE FILE
SRCHF   EQU     17              ;SEARCH FOR FIRST
ERASE   EQU     19              ;DELETE FILE
READ    EQU     20              ;READ FILE RECORD
WRITE   EQU     21              ;WRITE FILE RECORD
MAKE    EQU     22              ;CREATE NEW FILE
STDMA   EQU     26              ;SET DATA BUFFER POINTER
BDOS    EQU     0005H           ;SYSTEM I/O ENTRY POINT
FCB     EQU     5CH             ;SYSTEM FCB
FCBEXT  EQU     FCB+12          ;FILE EXTENT
FCBSNO  EQU     FCB+32          ;SECTOR #
FCB2    EQU     6CH             ;SECOND FCB
DSKBUF  EQU     080H            ;DEFAULT DISK BFFR ADDR
SECSIZ  EQU     080H            ;CP/M SECTOR SIZE
;
WBOOT   EQU     00              ;CP/M WARM BOOT ENTRY ADDR
;
;
;DEFINE ASCII CHARACTERS USED
;
LF      EQU     10              ;LINEFEED
CR      EQU     13              ;CARRIAGE RETURN
EOFCHR  EQU     01AH            ;CP/M END OF FILE CHAR
;
;
;START OF EXECUTABLE CODE
;
        ORG     100H
        LXI     SP,STACK  ;SETUP A STACK TO USE
;
;
;SEQUENTIAL I/O WRITE OF CP/M FILE ENABLED BY
;USING THIS SEQUENCE OF SUBROUTINE CALLS. THE FCB
;IS ASSUMED TO BE STORED AT THE DEFAULT LOCATION
;AT 05CH IN THE BASE PAGE OF CP/M MEMORY MAP.
;
SIOWR:
        CALL    ERASFIL   ;ERASE RECEIVED FILE
        CALL    MAKEFIL   ;ESTABLISH NEW FILE
        CALL    INITWR    ;INIT FILE WRITE
                          ;PARAMETERS
;
;MAKE FOLLOWING CALL TO PLACE A CHARACTER
;FROM THE (A) REGISTER INTO THE CP/M FILE. LOOP
;DOING THIS TILL YOU HAVE ALL IN FILE THAT
;IS NEEDED.
;
        CALL    WRCHAR    ;PUT CHAR IN FILE
;
        CALL    WREOF ;FLUSH LAST SCTR TO CP/M FILE
        CALL    CLOSFIL   ;CLOSE IT UP
;
;
;SEQUENCE OF COMMAND CALLS TO OPEN
;AND USE A SEQUENTIAL CHARACTER FILE FOR
;READING. THE FILE CONTROL BLOCK IS ASSUMED
;TO BE LOCATED AT DEFAULT LOCATION OF 05CH
;IN THE BASE CP/M PAGE. ONCE THE FILE IS
;INITIALIZED THE CHARACTERS CAN BE READ ONE BY
;ONE UNTIL THE RDCHAR SUBROUTINE RETURNS
;A SET CARRY FLAG INDICATING END OF PHYSICAL FILE
;CONDITION. EOF IS SENSED AS PHYSICAL END OR
;01AH CHARACTER WHICHEVER COMES FIRST
;
SIORD:
        CALL    OPENFIL   ;OPEN THE CP/M FILE
        CALL    INITRD    ;GO INIT FOR FILE READ
        CALL    RDCHAR    ;GET CHAR FROM CP/M FILE
        JC      EOF       ;CHECK FOR EOF
;
EOF:
;PLACE CODE HERE FOR END OF FILE HANDLING
;
;I/O HANDLING SUBROUTINES
;
;
;
;>-->    ERASFIL: ERASE THE INCOMING FILE.
;
;IF IT EXISTS, ASK IF IT MAY BE ERASED.
;
ERASFIL:
        LXI     D,FCB     ;POINT TO CTL BLOCK
        MVI     C,SRCHF   ;SEE IF IT..
        CALL    BDOS      ;..EXISTS
        INR     A         ;FOUND?
        RZ                ;..NO, RETURN
        CALL    ILPRT     ;PRINT:
        DB      ´++CP/M FILE EXISTS, TYPE Y TO ERASE: ´,0
        CALL    KEYIN     ;GET CHARACTER FROM CONSOLE
        ANI     5FH       ;MAKE UPPER CASE
        CPI     ´Y´       ;WANT ERASED?
        JNZ     EXIT      ;QUIT IF NOT ERASE
        CALL    CRLF      ;BACK TO START OF LINE
;
;ERASE OLD FILE
;
        LXI     D,FCB     ;POINT TO FCB
        MVI     C,ERASE   ;GET BDOS FNC
        CALL    BDOS      ;DO THE ERASE
        RET               ;FROM "ERASFIL"
;
;>-->    MAKEFIL: MAKES FILE TO BE RECEIVED
;
MAKEFIL:
        LXI     D,FCB     ;POINT TO FCB
        MVI     C,MAKE    ;GET BDOS FNC
        CALL    BDOS      ;TO THE MAKE
        INR     A         ;FF=BAD?
        RNZ               ;OPEN OK
;
;
;DIRECTORY FULL - CAN´T MAKE FILE
;
        CALL    ERXIT
        DB      ´++ERROR - CANNOT MAKE FILE´,CR,LF
        DB      ´++DIRECTORY MUST BE FULL´,CR,LF,´$´
;
```

```
;
;>-->    OPENFIL: OPENS THE FILE TO BE SENT
;
OPENFIL:
         LXI     D,FCB      ;POINT TO FILE
         MVI     C,OPEN     ;GET FUNCTION
         CALL    BDOS       ;OPEN IT
         INR     A          ;OPEN OK?
         RNZ                ;FILE OPENED OK
         CALL    ERXIT      ;..NO, ABORT
         DB      ´++CANNOT OPEN CP/M FILE´,´$´
;
;
;>-->    CLOSFIL: CLOSES THE RECEIVED FILE
;
CLOSFIL:
         LXI     D,FCB      ;POINT TO FILE
         MVI     C,CLOSE    ;GET FUNCTION
         CALL    BDOS       ;CLOSE IT
         INR     A          ;CLOSE OK?
         RNZ                ;..YES, RETURN
         CALL    ERXIT      ;..NO, ABORT
         DB      ´++CANNOT CLOSE CP/M FILE´,´$´
;
;
;>-->  INITRD: INITIALIZES FILE READ PARAMETERS
;
INITRD:
         MVI     A,00H      ;SET BUF CNT TO EMPTY
         STA     CHRINBF
         LXI     D,DSKBUF   ;SET DMA BUFFER POINTER
         PUSH    D
         MVI     C,STDMA
         CALL    BDOS
         POP     D
         XCHG               ;SET SECTOR POINTER
         SHLD    SECPTR
         RET
;
;
;>-->    RDCHAR: READS A CHARACTER FROM FILE
;
;RETURN IS WITH DESIRED CHARACTER IN
;THE A REGISTER. IF EOF, THEN
;RETURN IS WITH THE CARRY FLAG SET.
;
RDCHAR:
         LDA     CHRINBF    ;GET # OF CHAR IN BUF
         ORA     A          ;CHECK IF BUFFER EMPTY
         JZ      RDBLOCK    ;GO GET A SECTOR IF EMPTY
         DCR     A          ;DECREMENT
         STA     CHRINBF
         LHLD    SECPTR     ;GET BUFFER POINTER
         MOV     A,M        ;GET CHARACTER FOR CALLER
         INX     H          ;INCREMENT POINTER
         SHLD    SECPTR
         CPI     EOFCHR     ;CHECK FOR LOGICAL CP/M
         STC                ;EOF
         RZ                 ;RET EXIT FOR LOGICAL EOF
         CMC                ;CLR CARRY SO EOF NOT
                            ;INDICATED ON NORMAL RET
         RET                ;FROM "RDCHAR"
;
;
;BUFFER IS EMPTY - READ IN ANOTHER SECTOR
;
RDBLOCK:
         LXI     D,FCB
         MVI     C,READ
         CALL    BDOS
         ORA     A          ;READ OK?
         JZ      RDBFULL    ;YES
         DCR     A          ;EOF?
         JZ      REOF       ;GOT EOF
;
;
;READ ERROR
;
         CALL    ERXIT
         DB      ´++CP/M FILE READ ERROR´,´$´
;
REOF:
         STC                ;SET CARRY FLAG FOR EOF EXIT
         RET
;
;
;BUFFER IS FULL
;
RDBFULL:
```

```
         MVI     A,SECSIZ   ;INIT BUF CHAR COUNT
         STA     CHRINBF
         LXI     H,DSKBUF   ;INIT BUFFER..
         SHLD    SECPTR     ;..POINTER
         JMP     RDCHAR     ;PASS CHAR TO CALLER
;
;
;>-->  INITWR: INITIALIZES FILE WRITE PARAMETERS
;
INITWR:
         MVI     A,00H      ;SET THE BUF CNT
         STA     CHRINBF    ;TO EMPTY
         LXI     D,DSKBUF   ;SET THE DMA BUFFER
         PUSH    D          ;POINTER
         MVI     C,STDMA
         CALL    BDOS
         POP     D
         XCHG               ;SET SECTOR POINTER
         SHLD    SECPTR
         RET
;
;
;>-->    WRCHAR: WRITE A CHARACTER TO FILE
;
;ENTRY IS WITH CHARACTER IN A
;ENTRY AT WREOF FILLS REMAINING BYTES
;OF SECTOR WITH 01AH PER CP/M CONVENTION.
;
WRCHAR:
         LHLD    SECPTR     ;PUT CHAR IN BUFFER
         MOV     M,A
         INX     H          ;BUMP POINTER
         SHLD    SECPTR
         LDA     CHRINBF    ;INCR CHAR COUNT
         INR     A
         STA     CHRINBF
         CPI     SECSIZ     ;CHECK IF SECTOR FULL
         RNZ                ;GO BACK IF OK
;
WRBLOCK:
         LXI     D,FCB      ;IF FULL THEN WRITE
         MVI     C,WRITE    ;..THE..
         CALL    BDOS       ;..BLOCK
         ORA     A
         JNZ     WRERR      ;OOPS, ERROR
         MVI     A,00H      ;RESET THE CHAR CNT
         STA     CHRINBF
         LXI     H,DSKBUF   ;RESET BUFFER..
         SHLD    SECPTR     ;..POINTER
         RET
;
WRERR:
         CALL    ERXIT      ;EXIT W/MSG:
         DB      ´++ERROR WRITING CP/M FILE´,CR,LF,´$´
;
WREOF:
         LDA     CHRINBF    ;FILL REST OF SECTOR
         LHLD    SECPTR     ;WITH 01AH
         MVI     B,EOFCHR
WREND:
         MOV     M,B        ;PUT IN CP/M EOF CODE
         INX     H
         INR     A          ;INC THE CHAR CNT
         CPI     SECSIZ     ;BUFFER FULL YET
         JNZ     WREND
         JMP     WRBLOCK    ;GO PUT FILLED BLOCK
                            ;ON DISK
;
;>--> KEYIN: GETS A KEY CODE IN FROM CONSOLE
;
KEYIN:
         PUSH    B          ;SAVE..
         PUSH    D          ;..ALL..
         PUSH    H          ;..REGS
         MVI     C,RDCON    ;GET CON CHAR FNCTN CODE
         CALL    BDOS       ;GET CHARACTER
         MOV     A,E
         POP     H          ;RESTORE..
         POP     D          ;..ALL..
         POP     B          ;..REGS
         RET
;
;
;>-->    CTYPE: TYPES VIA CP/M SO TABS ARE EXPANDED
;
CTYPE:
```

(continued on next page)

```
        PUSH    B           ;SAVE..
        PUSH    D           ;..ALL..
        PUSH    H           ;..REGS
        MOV     E,A         ;CHAR TO E
        MVI     C,WRCON     ;GET BDOS FNC
        CALL    BDOS        ;PRIN THE CHR
        POP     H           ;RESTORE..
        POP     D           ;..ALL..
        POP     B           ;..REGS
        RET                 ;FROM "CTYPE"
;
;
;>--> CRLF: TYPE CARRIAGE RET LINE FEED PAIR
;
CRLF:
        MVI     A,CR
        CALL    CTYPE
        MVI     A,LF
        CALL    CTYPE
        RET
;
;
;>-->    ILPRT: INLINE PRINT OF MSG
;
;THE CALL TO ILPRT IS FOLLOWED BY A MESSAGE,
;BINARY 0 AS THE END.  BINARY 1 MAY BE USED TO
;PAUSE (MESSAGE ´PRESS RETURN TO CONTINUE´)
;
ILPRT:
        XTHL                ;SAVE HL, GET HL=MSG
ILPLP:
        MOV     A,M         ;GET CHAR
        ORA     A           ;END OF MSG?
        JZ      ILPRET      ;..YES, RETURN
        CPI     1           ;PAUSE?
        JZ      ILPAUSE     ;..YES
        CALL    CTYPE       ;TYPE CHARACTER OF MESSAGE
ILPNEXT:
        INX     H           ;TO NEXT CHAR
        JMP     ILPLP       ;LOOP
;
;
;PAUSE WHILE TYPING HELP SO INFO DOESN´T
;SCROLL OFF OF VIDEO SCREENS
;
ILPAUSE:
        CALL    ILPRT       ;PRINT:

        DB      CR,LF,´PRESS RET TO CONT OR ^C TO EXIT
        DB      CR,LF,0
        CALL    KEYIN       ;GET ANY CHAR
        CPI     ´C´-40H     ;REBOOT?
        JZ      EXIT        ;YES.
        JMP     ILPNEXT     ;LOOP
;
ILPRET:
        XTHL                ;RESTORE HL
        RET                 ;& RET ADDR PAST MESSAGE
;
;
```

```
;>-->    PRTMSG: PRINTS MSG POINTED TO BY (DE)
;
;A ´$´ IS THE ENDING DELIMITER FOR THE PRINT.
;NO REGISTERS SAVED.
;
PRTMSG:
        MVI     C,PRINT     ;GET BDOS FNC
        JMP     BDOS        ;PRINT MESSAGE, RETURN
;
;
;>-->    ERXIT: EXIT PRINTING MSG FOLLOWING CALL
;
ERXIT:
        POP     D           ;GET MESSAGE
        CALL    PRTMSG      ;PRINT IT
;
EXIT:
        LXI     D,080H      ;RESET DEFAULT DMA
        MVI     C,STDMA     ;ADDRESS FOR EXIT
        CALL    BDOS
        LHLD    STACK       ;GET ORIGINAL STACK
        SPHL                ;RESTORE IT
        JMP     WBOOT       ;GO DO CP/M WARM BOOT TO
                            ;BRING BACK IN CCP
;
;
;FOLLOWING 2 USED BY THE CP/M
;DISK BUFFERING ROUTINES
;
SECPTR  DW      DSKBUF      ;POINTER TO DISK BUFFER POS
CHRINBF DB      0           ;# OF CHARACTERS IN BUFFER
;
;
;SETUP A STACK AREA
;
        DS      38          ;STACK AREA
STACK   DS      2           ;STACK POINTER
;
;   --------------
;
        END
;
;+++...END OF LISTING 2
```

You're invited to join us again next month when the tutorial continues into its third and final part. The functions of random record file I/O will be presented with complete programming examples to show how random I/O works. Several special file I/O tricks will be shown that permit unique problems to be solved under the CP/M operating system. One of these will be a program that performs "update" on an existing file without the use of the random record I/O capabilities. So long till February and I hope that all *Lifelines/The Software Magazine* readers have a joyous holiday season. ■

Feature 8080 Assembler Programming Tutorial, Macros

Ward Christensen

I program in assembler because I like the concise and efficient programs that can be written in it. Structured programming, which I mentioned in an earlier tutorial, can help you organize your thoughts and ease the programming task. There are also other ways.

High level languages provide a means of writing programs where the *writing* is more efficient, although at execution you pay a speed and size penalty. Many and perhaps most applications can justify this expense because of the convenience of programming in a high level language.

There is an alternative: macros. In plain English, the word means "large". In computer terms, it is short for "macroinstruction". It refers to generating many instructions from a single one. Digital Research's MAC is a macro assembler, selling for under $100. It has a nice instruction manual which includes many examples. RMAC is DR's newer macro assembler, generating relocatable routines which may later be linked together to form an executable COM file.

Aside: althoughI have purchased RMAC, I have not had time to get familiar with it. However, if you are considering buying a macro assembler, you might consider RMAC over MAC, so you'll be ready for an upcoming CPMUG project suggested by Dean Dwyer — a library of macros and subroutines to facilitate efficient programming using RMAC.

In this month's tutorial, I will present an overview of how macros work, and give some practical examples which I use in my everyday programming.

OVERVIEW of MACROS

"MACROS GENERATE ASSEMBLY INSTRUCTIONS"

Macros have the ability to do many things: arithmetic, counting, scanning, substituting, etc. All these are done for the sake of generating assembly instructions.

The most basic type of macro simply generates a fixed series of instructions. For example, you could write a macro called "EXIT" which you code in a CP/M program to restore the stack and return. It generates:

```
        lhld    stack   ;get stack
        sphl            ;restore it
        ret             ;ret to CP/M
```

The code necessary to generate this sequence of instructions by using the EXIT macro is:

```
exit    macro

        lhld    stack   ;get stack
        sphl            ;restore it
        ret             ;ret to CP/M
        endm
```

The pseudo-operation "macro" tells MAC that a macro definition follows. The label "exit" defines the name of the macro. "Endm" tells MAC that the macro had ended.

Where do macros come from?

Macro *definitions* may be placed in the program itself, or in a file of macros (usually called a library) e.g., "name.LIB". You call for the library at the beginning of your assembly source program by coding:

```
        MACLIB name
```

When experimenting with macros it is easiest just to code the definitions at the front of the program which uses them.

Practical Macros

My favorite macro is one to interface to BDOS. In CP/M programs, I frequently code the sequence:

```
    mvi     c,..function..
    call    bdos
```

or

```
    lxi     d,..something..
    mvi     c,..function..
    call    bdos
```

or, if I want my registers saved, this becomes:

```
    push    b
    push    d
    push    h
    lxi     d,..something..
    mvi     c,..function..
    call    bdos
    pop     h
    pop     d
    pop     b
```

Those 9 instructions are easy to code, but become tedious when you code them over and over, with only minor variations. Macros to the rescue!

What would a macro to implement these BDOS have to do: It should always generate the "call bdos"; it should optionally load C with the function to be passed (OPEN, SETDMA, etc); it should optionally load DE with the parameter to be passed (FCB address, etc); and it should optionally save and restore the registers.

Let's look at a typical program of mine, CLIST.ASM, which makes use of this macro. It is a general purpose listing program, originally for my Centronics printer, (thus the "C" in CLIST.ASM). CLIST uses this macro, which I called "CPM", 14 times.

There are three operands on the macro. You may code any combination of them. The operands are *positional*, meaning MAC detects them by their position: first, second, or third. If you omit an operand but code a later one, you *must* show the omitted ones by an appropriate number of commas. More about this later.

My CPM macro operands are:

```
    CPM    function,parameter,nosave
```

"function" is a BDOS function, to be placed in the C register, OPEN, SETDMA, etc. The actual values of these functions come from EQU statements at the end of my program: OPEN EQU 15 for example.

"parameter" is what is to be placed in E or DE, such as the FCB address.

"nosave" is specified as any character string, and causes the register saves to be omitted. I usually just code the word NO.

The calls themselves, taken out of context of CLIST.ASM are things like:

```
        CPM     OPEN,FCB
        CPM     CONST
        CPM     RDCON
        CPM     SETDMA,80H
        CPM     SRCHF,FCB
        CPM     WRCON,CR,NO
        CPM     WRCON,LF,NO
        CPM     WRCON
        CPM     READ,FCB
        CPM     SETDMA,80H
```

If I had wished to not save the registers in the call to console status (CONST), I would have had to code:

```
        CPM     CONST,,NO
```

with the commas showing an omitted second operand. Similarly, if I already had the BDOS function in C, and the parameter in DE or E, I could just code the BDOS call with register saves as:

```
        CPM     ...NO
```

Writing Macros

The CPM macro itself begins with the line:

```
CPM     MACRO   ?F,?P,?N
```

The operation code MACRO tells MAC this is a macro. CPM gives it a name, the the ?F, ?P, and ?N tell MAC it may have up to three operands. ("?" is a valid label character to MAC, and I use it to designate the parameters.)

The operands you code when you execute the macro are internally assigned to the corresponding operands on the MACRO command. Thus if you were to code:

```
        cpm     setdma
```

MAC would store the character string "setdma" as the current value of ?F. Thus if the macro definition contains:

```
        MVI     C,?F
```

it will become:

```
        MVI     C,setdma
```

when the macro is actually executed expanded into assembler source by MAC.

Macros frequently make use of the assembler IF and ENDIF to decide if certain instructions are to be generated or not. If the operand of the IF statement evaluates to a non-zero value, the instructions up to the next ENDIF are generated. If the operand is false, no instructions are generated.

MAC supports more tests for use in IF than ASM does. One used by my CPM macro is "NUL". It tests the operand to see if it has been coded. Thus, if the CPM macro were coded with no operands, ?F would have no (a "null") value. The macro line:

```
        IF      NUL ?F
```

(continued on next page)

tests to see if the operand was coded. With that background, here is the entire macro:

```
CPM     MACRO   ?F,?P,?N
        IF      NUL ?N
        PUSH    B
        PUSH    D
        PUSH    H
        ENDIF
        IF      NOT NUL ?F
        MVI     C,?F
        ENDIF
        IF      NOT NUL ?P
        LXI     D,?P
        ENDIF
        CALL    BDOS
        IF      NUL ?N
        POP     H
        POP     D
        POP     B
       ENDIF
       ENDM
```

Going through it: "IF NUL ?N" tests to see if the third operand was coded. Recall the third operand is coded if you want to suppress the PUSH and POP register saves. Thus, if "?N" is NUL, the following instructions: PUSH B, PUSH D, and PUSH H, are not generated. ENDIF ends the "IF NUL ?" test.

Similarly, "IF NOT NUL ?F" tests to see if a function, such as OPEN or CLOSE was coded, and if so, generates "MVI C, ?F". Again, ENDIF ends the test.

"IF NOT NUL ?P" tests to see if a parameter is coded, generating "LXI D,?P" if so. ENDIF again balances the "IF" statement. Note that single-byte values may be coded for the parameter, such as 0dh. LXI loads an 8-bit value by putting 00 in D, and the 8-bit value in E, so an LXI may be used whether or not an 8- or 16-bit operand is coded.

The "CALL BDOS" is then generated. Then, "IF NUL?N" again tests if the registers were to be saved and generates the POPs if so. ENDIF ends the generation of POPS. Finally "ENDM" ends the macro definition.

When MAC sees this macro definition, it does not execute it immediately, but rather loads it into the symbol table, where MAC can later find it when it encounters a call to the macro such as:

The size of macros are limited, since they must be loaded into the symbol table. Comments in macros, if specified by ";;" instead of ";", are NOT loaded into the symbol table.

Sample Program

Assume the CPM macro has been edited into a file called "TEST.LIB". The following sample program uses the CPM macro. The program tests to see if a file exists, typing the phrases "found" or "Not found" as appropriate. While not a very useful program (DIR is better), it does show how a very few lines of code can interface to BDOS.

LISTING 1 shows the .PRN file from the assembly with MAC. Lines which have a " + " in the column between the address and the object code are lines generated by macros. Before the first one in each series is the CPM macro that caused the code to be generated. Note that the CPM macro line itself doesn't generate any object code.

More Macros

There are other macros that I frequently use, such as MOVE and COMP. Since an 8080 microprocessor doesn't directly support the moving of a block of data from one location to another, a macro may be written to do this for you. The COMP macro similarly compares two character strings in memory. Here are some sample executions of the macros:

```
comp    'ASM',fcb+9
move    'TEST    BAK',fcb
move    '0001',lineno
move    fcb2,myfcb,33
```

LISTING 2 shows the definition of these macros, and LISTING 3 shows the actual MOVER and COMPR subroutines that are called by the macros.

The routines consist of two pieces: (1) the macro issuing the call, and (2) the actual subroutine that "does the work". The Digital Research MAC manual shows how to define macros for calling such subroutines. It "redefines" the macros in such a way as to generate the subroutine the first time the macro is called, then redefine the macro to only generate the register loads and the call to the subroutine from then on. I "think" a bit different than that, preferring to have the subroutine at the end of the program. All I do is set a switch at the front of the program, namely MF for move flag, and CF for compare flag, to 0. Then if I issue any MOVE or COMP macros, the appropriate flag is set true. At the end of the program (see listing 2) I test the flags to see if it is necessary to generate the subroutine code.

The MOVE and COMP macros are so similar — namely the first operand is loaded into HL, the second into DE, and the length into BC, that I decided not to code duplicate tests for these operands, but instead to define an *inner macro*, i.e., like a macro subroutine, that is called by MOVE and COMPR macros. I called this macro "MCSUB", for Move and Compare SUBroutine.

MCSUB (see LISTING 2) is quite complex, and shows the power of MAC macros. Here is what it does:

I first test to see if the "from" or "first" operand is coded, with:

```
        IF      NOT NUL ?F
```

This means the following instructions will be executed if the "from" field, designated ?F, is not nul, i.e., if it exists. It first executes:

```
        IRPC    ?C,?F
```

which is a "built-in" macro of MAC or RMAC. It stands for "indefinite RePeat Character", and loops, stepping character-by-character through operand ?F, placing each subsequent character in ?C. I am doing this to set up a test for the first character being a quote. This allows the "from" field in a move or compare to be a character literal. The next statement:

```
?Q      SET     '&?C&?C' ;;TEST FOR QUOTE
```

makes use of another ability of macros: to substitute single characters, even in quotes. A '&' says to take the next parameter literally. Thus &?C is changed to the character value of ?C, and '&C&?C' becomes 'xx' where 'x' is the value of ?C.

The reason I set *two* values, is that in assembly language, a single quote is used to delimit a character literal, so two consecutive quotes are used to represent a single quote. Thus the expression ''' is actually invalid, since the first quote is taken to open the string, the second two represent a single quote, then there is no closing quote.

Next I code:

```
        EXITM
        ENDM
```

The EXITM causes the IRPC to end. Normally, you would loop through it until there are no more characters.

However, I only wanted to execute it once to test for the quote. Thus, EXITM exits the macro immediately. ENDM formally terminates the IRPC macro.

Now it's time to test ?Q to see if it is a quote:

```
                IF       ?Q EQ ''''
```

If ?Q is a quote, I want to generate the following sort of code:

```
        CALL     XXX
DDD     DB       'THE REQUESTED LITERAL'
XXX     POP      H
        LXI      B,XXX-DDD
```

The CALL sets up DDD as a return address, i.e., points the value on the top of the stack to DDD. However, the POP H at XXX pops that pointer to DDD into HL. Thus I have satisfied the requirements of the move and compare subroutines — pointing HL to the literal. The B index register must contain the length of the move or compare, so LXI B computes the length by subtracting the labels, and loading the result to B.

Here's how that is coded in the macro:

```
        LOCAL    ?B,?Z
        CALL     ?Z
?B      DB       ?F
?Z      POP      H         ;GET FROM
        LXI      B,?Z-?B ;GET LEN
```

LOCAL is a special pseudo-operation to MAC that tells it to make unique values for ?B and ?Z each time the macro is executed. If this weren't done, multiple executions of the macro would cause duplicate labels to be generated. Thus ?B in the first macro actually becomes "??0001", ?Z becomes "??0002", and in the second macro they become "??0003" and "??0004".

Now, we get to another ability of MAC not shared by ASM: handling an ELSE as part of an IF. The IF was: IF ?Q EQ '''', so now the ELSE is handling the case where ?F did not start with a quote:

```
        ELSE
        LXI      H,?F
        ENDIF
        ENDIF
```

Note that IF/ENDIF may be nested, again unlike ASM. The first ENDIF ended the "IF ?Q EQ '''' ", and the second ended the "IF NOT NUL ?F".

Finally:

```
        IF       NOT NUL ?T
        LXI      D,?T
        ENDIF
        IF       NOT NUL ?L
        LXI      B,?L
        ENDIF
```

tests for the ?T and ?L operand, loading them into DE and BC respectively. Then:

```
        ENDM
```

ends the MCSUB macro. Control returns to the MOVE or COMP macro, which then generates the appropriate call to MOVER or COMPR. Thus endeth the macro.

I have attempted to give you a bit of the flavor of macros, to help you decide if you are interested in using them to make your assembler programming more efficient.

Incidentally, unlike ASM, MAC outputs a symbol table of all the labels in an assembly. This, when used with Digital Research's Symbolic Instruction Debugger, provides very good productivity in solving troublesome program bugs. SID is also under $100, and a very good buy.

```
                        ;
                        ; SAMPLE PROGRAM USING CPM MACRO.
                        ;
                                  MACLIB  TEST
                        ;
0100                              ORG     100H              ;START CODE HERE
0100 210000                       LXI     H,0               ;HL = 0
0103 39                           DAD     SP                ;HL = CCP'S STACK
0104 229C01                       SHLD    STACK             ;SAVE CCP'S STACK
0107 319C01                       LXI     SP,STACK          ;LOAD OUR STACK
                                  CPM     SRCHF,FCB,NO      ;SEARCH FOR THE FILE
010A+0E11                         MVI     C,SRCHF
010C+115C00                       LXI     D,FCB
010F+CD0500                       CALL    BDOS
0112 3C                           INR     A                 ;IF OFFH MAKE IT 0
0113 C22101                       JNZ     FOUND             ;IF NOT NOW 0, THEN FOUND
                                  CPM     PRINT,NOTMSG,NO ;PRINT "NOT FOUND" MSG.
0116+0E09                         MVI     C,PRINT
0118+112E01                       LXI     D,NOTMSG
011B+CD0500                       CALL    BDOS
011E C32901                       JMP     EXIT              ;RETURN TO CCP
                        ;
                        FOUND     CPM     PRINT,MSG,NO      ;PRINT "FOUND" MSG.
0121+0E09                         MVI     C,PRINT
0123+113201                       LXI     D,MSG
0126+CD0500                       CALL    BDOS
0129 2A9C01             EXIT      LHLD    STACK             ;GET CCP'S STACK
012C F9                           SPHL
012D C9                           RET                       ;RETURN TO CCP
                        ;
012E 4E6F7420           NOTMSG    DB      'Not '            ;1ST PART OF "NOT FOUND"
0132 666F756E64MSG                DB      'found$'          ;"FOUND" MESSAGE
                        ;
0138                              DS      100               ;STACK SPACE
019C                    STACK     DS      2                 ;SAVE STACK HERE
                        ;
                        ;EQUATES USED IN PROGRAM:
                        ;
0005 =                  BDOS      EQU     5                 ;BDOS ENTRY ADDR
005C =                  FCB       EQU     5CH               ;FILE CONTROL BLOCK
0009 =                  PRINT     EQU     9                 ;BDOS PRINT MSG TO '$'
0011 =                  SRCHF     EQU     17                ;SEARCH FOR FILE
019E                              END
```

Listing 1

MAC output

```
----------------------------------------------------------------------

MF      SET     0       ;SHOW MOVE NOT REQUESTED
CF      SET     0       ;SHOW COMP NOT REQUESTED
;
;---->  MOVE from,to,length
;       from may be addr, or quoted string
;
MOVE    MACRO   ?F,?T,?L
        MCSUB   ?F,?T,?L ;;HANDLE ARGS
        CALL    MOVER
MF      SET     -1      ;;SHOW EXPANSION
        ENDM
;
;COMPARE MACRO
COMP    MACRO   ?F,?T,?L
        MCSUB   ?F,?T,?L ;;HANDLE ARGS
        CALL    COMPR
CF      SET     -1      ;;SHOW EXPANSION
        ENDM
;
;MCSUB - HANDLES MOVE, COMPARE ARGUMENTS
MCSUB   MACRO   ?F,?T,?L
        IF      NOT NUL ?F
        IRPC    ?C,?F
?Q      SET     '&?C&?C' ;;TEST FOR QUOTE
        EXITM
        ENDM
        IF      ?Q EQ ''''
        LOCAL   ?B,?Z
        CALL    ?Z
?B      DB      ?F
?Z      POP     H       ;GET FROM
        LXI     B,?Z-?B ;GET LEN
        ELSE
        LXI     H,?F
        ENDIF
        ENDIF
        IF      NOT NUL ?T
        LXI     D,?T
        ENDIF
        IF      NOT NUL ?L
        LXI     B,?L
        ENDIF
        ENDM
```

Listing 2

MOVE, COMP macros

```
----------------------------------------------------------------------
;
;MOVE, COMPARE SUBROUTINES
;
        IF      MF      ;MACRO EXPANSION FLAG SET?
MOVER   MOV     A,M     ;get a byte
        STAX    D       ;store in output field
        INX     H       ;bump input pointer
        INX     D       ;bump output pointer
        DCX     B       ;decrement byte count
        MOV     A,B     ;get high byte count
        ORA     C       ;"or" with low
        JNZ     MOVER   ;loop if BC not yet 0
        RET             ;       otherwise return
        ENDIF
;
        IF      CF      ;MACRO EXPANSION FLAG SET?
COMPR   LDAX    D       ;get byte from first field
        CMP     M       ;compare to second field
        RNZ             ;return if unequal match
        INX     D       ;otherwise bump
        INX     H       ;       the two pointers
        DCX     B       ;       and decrement count
        MOV     A,B     ;       until
        ORA     C       ;       count = 0
        JNZ     COMPR   ;       loop if not
        RET             ;otherwise return
        ENDIF
```

Listing 3

MOVER, COMPR subroutines

Figure 3. FILE CONTROL BLOCK DESCRIPTION

dr	f1	f2	/ /	f8	t1	t2	t2	ex	s1	s2	rc	d0	/ /	dn	cr	r0	r1	r2
00	01	02	. . .	08	09	10	11	12	13	14	15	16	. . .	31	32	33	34	35

where:

dr drive code (0 - 16)
0 => default drive for file access
1 => select drive A: for file access
2 => select drive B: for file access
.
16 => select drive P: for file access

f1 . . . f8 contain the file's name in ASCII upper case with high bits equal to zero

t1,t2,t3 contain the file type in ASCII upper case; high bits normally equal zero. tn′ indicates the high bit of these positions.
t1′ = 1 => Read/Only file
t2′ = 1 => SYS file, no DIR list

ex contains current extent number, normally set to 00, but in the range 0 - 31 during file I/O.

s1 for internal system use

s2 for internal system use, set to zero on call to OPEN, MAKE, SEARCH system calls.

rc record count for extent "ex," can assume values 0 to 128.

d0 . . . dn filled-in by BDOS to tag file group numbers for this extent.

cr current record to read or write in sequential file operation. User normally sets this on initial access to file.

r0,r1,r2 optional random record number ranging from 0 to 65535, with overflow to r2. r0/r1 are 16 bit value in low/high byte order.

New Products

SAPANA-EXPENSE-TRACK-I

Sapana Micro Software

This is a menu-driven, easy-to-use program for keeping track of expenses in a home or small business. Some features of Expense-Track-I are: it allows several expense files on a diskette; each entry has a date, description, category code, method of payment code, tax status and expense items, all of which are checked for validity before they are accepted. In addition, they can all be printed on screen and/or printer.

The program is priced at $29, with manual. It requires an IBM PC under DOS, a printer, 64K memory, one disk drive.

FILE PRO

The small Computer Company, Inc.

This is a CP/M-80 data base management software package for the NEC PC-8000A personal computer. It is fully menu-driven and takes advantage of the PC-8000A's graphics, color, blinking, reverse video, shading, underscoring and five function keys. Using a full screen editor and function keys, the designer can interactively create customized menus for operators, lay out up to five input screens and create up to five prestored report and label formats.

FilePro is capable of storing 64,000 records (with hard disk) 99 fields per record, and 1020 characters per record. It can perform high-speed searches, using any two of up to 36 fields. It also allows for associated and computed fields and mass update and recalculation of records.

It requires a NEC PC-8000A, CP/M-80, 64K RAM and two drives. Cost is $199.

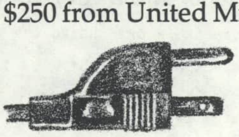
TAXCUT

United Micro Systems, Inc.

TaxCut is a new software program which allows users to test outcomes of hundreds of tax related alternatives. When the data is entered, the taxpayer can print out the complete return – including many schedules which will be ready to sign and mail.

Examples of tax-related decisions TaxCut can handle are IRA's, Keogh plans, and investment in business equipment. The package will project figures for income averaging, installment sales, investment credit, etc. Provisions of the latest laws, all-savers certificates, charitable contributions for those who don't itemize are incorporated.

TaxCut is a menu-driven system and includes user-oriented documentation. The TaxCut software was designed for the IBM PC and is currently being converted to the Apple II and other computers operating SB-80 or CP/M-80. The TaxCut diskette is available by mail-order for $250 from United Micro Systems.



"...Lattice C compiler is the best compiler all around that I have ever seen outside of the UNIX environment. The quality and completeness... is truly awesome."

Jason T. Linhart (*Mark of The Unicorn,* Author of Mince)

In praising Lattice C, Jason Linhart has a lot of company. It is considered by experienced users to be the definitive compiler. This 8086/8088 C Compiler supports the full C language. It is not a subset. Lattice C takes advantage of the features provided by the 16-bit 8086 instruction set and is especially suitable for applications where clear structure is crucial.

Applications of considerable complexity and power can be developed—text processing, file manipulation, data modeling, system maintenance, and much more.

Lattice C accepts source code files written in C and produces relocatable machine code in Intel's™ 8086 object module format, which can be linked together into larger programs. The Lattice C library defines a comprehensive set of I/O subroutines that implement UNIX™-compatible standard functions.

Lattice C is ideal for anyone who wants to work with or learn C—for experienced programmers who wish to enjoy the clarity and speed of C in their applications; for anyone who wants the programming capabilities of a higher level language without sacrificing program efficiency. In fact, all of the program examples listed in *The C Programming Language* by Kernighan and Ritchie can be compiled by Lattice C.

Lattice C implements the C language on all Intel 8086/8088 code-compatible microcomputers, including the IBM™ PC under DOS, MS™-DOS, and SB-86™.

For more information about Lattice C and other programs available for the IBM PC and other 8086/8088 computers, just fill out the coupon or give us a call at (212) 860-0300.

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

Please send me further information on
□ Lattice C □ 16-bit software □ 8-bit software

Name ____________ Title ____________

Company ____________ Bus. Phone ____________

Street ____________

City ____________ State ____________ Zip ____________

Lifeboat Associates
World's No. 1 source of micro software

Yes, we're the biggest.

But that isn't what makes us number one.

It's the totality of what we do to make microcomputers more effective for you that makes us number one.

Yes. We have the largest number of packages—simple and complex. Yes. We have the greatest number of formats. Yes. We have the best technical support in the business. Yes. All of our products are immediately available.

But let's take a step back. When the microcomputer world opened up there was little definition and no software. Then came Lifeboat—to meet the need for easy-to-use, fully-tested, reliable software backed by extensive and available service.

Lifeboat developed standards for the industry which led to improved quality, reduced costs, higher levels of technical competence, credibility and reduced user risk.

Today Lifeboat offers personal, professional and corporate end-users, as well as dealers, distributors, authors, OEMs and others, a unique, single-source, full-service Software Support System.™

Everyone looks to us as the source of the most comprehensive, fully-tested line of software. Word processing, financial planning, accounting, graphics, data base management, languages and more. We have it all—for nearly every microcomputer available, including the IBM PC.

Our customer service department provides facilities for mail, telephone, TWX, telex and personal sales. We have a network of offices in the U.S., England, France, Japan, Switzerland and West Germany.

We provide a Software Desk Reference™ which contains up-to-date information about state-of-the-art software books, periodicals and accessories.

We offer subscriptions to *Lifelines*™ The Software Magazine.™ A monthly publication covering new products, tips for microcomputer users, product comparisons and other features to guide the reader before and after a purchase.

As the largest publisher of software, we also print a guide setting standards for software authors.

It takes a lot to become big but it takes even more to become—and remain—number one.

That's our commitment.

Lifeboat Associates, 1651 Third Avenue, New York, New York 10028.
TWX: 710-581-2524 (LBSOFT NYK). TELEX: 640693 (LBSOFT NYK).

□ Please send me a free Lifeboat Software Desk Reference filled with descriptions of over 200 programs designed for use in professional, business, programming and personal environments.

□ Please send further information on *Lifelines* and Author's Guide.

Name ____________________ Title ____________________

Company ____________________ Bus. Phone ____________________

Street ____________________

City ____________________ State ____________________ Zip ____________________

Lifeboat Associates
World's No. 1 source of micro software

Lifelines, The Software Magazine,™ Lifelines Pub. Co.
Software Support System, Software Desk Reference,™ Lifeboat Assoc.

NEW 16-Bit Software Available for the IBM PC, plus...

System Tools:
Emulator/86
EM80/86
PMATE-86
UT86
PANEL-86

Telecommunications:
ASCOM

Languages:
Lattice C Compiler
PL/M

Word Processing Systems And Aids
WordStar
MailMerge
MicroSpell
Spellguard

Data Management Systems:
T.I.M. III

Mailing List Systems
Postmaster

Financial Accounting Packages
General Ledger

Numerical Problem-Solving Tools
Math PC
Plan86
SigmaCalc
Statpak

Professional And Office Aids
Dental Mngmnt Sys. (8000 & 9000)
Insurance Agency
Legal Time Acctng.
Medical Mngmnt Series (8000 & 9000)

Disk Operating Systems:
MS-DOS (SB-86) — available for OEM license.

8-Bit Software Available

System Tools:
BUG and uBUG
DESPOOL
DISILOG
DISTEL
EDIT
EDIT-80
FILETRAN
IBM/CPM
MAC
MACRO-80
MINCE
PANEL
PASM
PLINK
PLINK II
PMATE
RAID
Reclaim
SID
TRS-80 Model II Cust. Disk
Unlock
WordMaster
XASM: 05, 09, 18, 48, 51, 65, 68, 75, F8, 400, Z8
ZAP80
ZDT
Z80 Development Package
ZSID

Telecommunications:
ASCOM
BSTAM
BSTMS
eZmail
MicroLink-80
RBTE-80

Languages:
ALGOL-60
APL/V80
BASIC Compiler
BASIC-80
baZic II
BD Software C Compiler
CBASIC-2
CIS COBOL (Standard)
COBOL-80
FORTRAN-80
KBASIC
JRT Pascal
muLISP/muSTAR
Nevada COBOL
Pascal/M
Pascal/MT
Pascal/M +
Pascal/Z
PL/I-80
Precision BASIC
STIFF UPPER LISP
S-BASIC
Timin FORTH
Tiny-C
Tiny-C TWO
UCSD Pascal
Whitesmiths' C Compiler
XYBASIC

Language and Applications Tools:
BASIC Utility Disk
DataStar
FABS
FABS II
Forms 2 for CIS COBOL
MAG/sam3,4
MAG/sort
M/SORT for COBOL 80
Programmer's Apprentice
PSORT
QSORT
STRING/80
STRING BIT
SuperSort
ULTRASORT II
VISAM

Word Processing Systems and Aids:
Benchmark
DocuMate/Plus
Letteright
MagicPrint
Magic Wand
Math★
MicroSpell
SMARTKEY
Spellguard
TEX
Textwriter III
WordIndex
WordStar
WordStar French
WordStar Customization Notes

Data Management Systems:
CONDOR
dBASE II
Formula
HDBS
Hoe
MAG/base1,2,3
MDBS
MicroSEED
T.I.M. III

General Purpose Applications:
CBS
CBS Label Option Pak
Selector III-C2
Selector IV

Mailing List Systems:
Benchmark Mailing List
Mailing Address
MailMerge for WordStar
NAD
Postmaster

Financial Accounting Packages:
BOSS Financial Accounting System
Financial Pkgs. (PTree)
Financial Pkgs. (SSG)
General Ledger Acctng (Univair)
GLector

Numerical Problem-Solving Tools:
Analyst
fpl
Microstat
muSIMP/muMATH
PLAN80
SigmaCalc
Statpak
T/MAKER II

Professional And Office Aids:
Apartment Mngmnt (Cornwall)
Datebook
Dental Mngmnt (Univair)
Dental Mngmnt-Family (Univair)
GrafTalk
Insurance Agency Mngmnt
Legal Time Acctng (Univair)
Medical Mngmnt (Univair)
Medical Mngmnt-Family (Univair)
PAS 3 Medical
PAS 3 Dental
Professional Time Acctng (PTA)
Property Mngmnt Pkg. (Am. Soft.)
Property Management (PTree)
Sales Pro
Wiremaster

Lifeboat After Hours
Backgammon/Gomoku

Educational Tools
Torricelli Author
Torricelli Studio

Books and Periodicals
APL—An Interactive Approach
Accounts Payable and Accounts Receivable-CBASIC
CBASIC User Guide
The Computer Glossary
The CP/M Handbook (with MP/M)
The C Programming Language
Crash Course in Microcomputing
Devil's DP Dictionary
Discover FORTH
DON'T (Or How To Care For Your Computer)
8080/Z80 Assembly Language Techniques For Improved Programming
Executive Computing
Fifty BASIC Exercises
General Ledger-CBASIC
Introduction to Pascal
Lifelines/The Software Magazine
Pascal User Manual and Report
The Pascal Handbook
The Pascal Primer
Payroll with Cost Accounting —CBASIC
Structured Microprocessor Programming
A User Guide To The UNIX System
Using CP/M—A Self-Teaching Guide

Hardware and Accessories
DC Data Cartridges
Diskette Drive Head Cleaning Kits
Flippy Disk Kit
Floppy Saver
Smartmodem
Vari Clean Cleaning Kit

Disk Operating Systems
BRIDOS
CP/M-80
MP/M
SB-80
APPLI-CARD
Softcard

Hard Disk Integration Modules

Media & Formats for 8-AND 16-Bit Microcomputers

This list of available formats is subject to change without notice. If you do not see your computer listed or are uncertain, call to confirm the format code for any particular equipment.

A.B. Dick	M8
ADDS Multivision	RT
AES Super Plus IV	Q4
ALSPA 8″	A1
Altair 8800	B1
Altos	A1
Apple CP/M-80 13 Sector	RG
Apple CP/M-80 16 Sector	RR
Archives 1	SG
AVL Eagle I	RB
AVL Eagle II	ST
BASF System 7100	RD
Blackhawk Micropolis Mod II	Q2
BMC iF-800	SR
Cado	A1
California Computer Sys 8″	A1
CDS Versatile 3B	Q1
CDS Versatile 4	Q2
Columbia Data Products 8″	A1
Columbia Data Products 5¼″	S4
Commodore CBM/PET + SSE Box + 8050	C2
Commodore CBM/PET w/Madison Z-RAM + 8050	C4
COMPAL-80	Q2
Compucorp 655	Q7
Compucorp 685	Q6
Computer Ops N.C. HQ	S2
Control Data 110	A1
CPT 8000	A1
Cromemco System 3	A1
Cromemco System 2 SD/SS	R6
Cromemco System 2 DD/SS	RX
Cromenco System 2 DD/DS	RY
CSSN Backup	T1
Datapoint 1550/2150 DD/SS	AA
Datapoint 1550/2150 DD/DS	AB
Datavue DU 80-222	M7
DEC VT 18 X	SD
Delta Systems	A1
Digi-Log Microterm II	RD
Digi-Log Sys. 1000/1500/2000	RD
Direct OA1000	M2
DTC Micro 210A	SC
Durango F-85	RL
Dynabyte DB8/2	R1
Dynabyte DB8/4	A1
Exidy Sorcerer + LB CP/M-80 5¼″	Q2
Exidy Sorcerer + Exidy CP/M-80 5¼″	RW
Exidy Sorcerer + Exidy CP/M-80 8″	A1
EXO	A1
Exxon 510/520	Q5
Findex	P6
Godbout	E1
Heath H8 + H47	A1
Heath H89 + Magnolia CP/M-80	P7
Heath H89 + Heath CP/M-80	P7
Helios II	B2
Heurikon MLZ, SS	SN
Heurikon MLZ, DS	SO
Heuristics HCC Spectrum	A1
Hewlett-Packard-87	SB
Hewlett-Packard 125, 5¼″	SB
Hewlett-Packard 125, 8″	A1
IBEX 7100	RQ
IBM Personal Computer	G1
ICL Personal Computer	RE
iCOM 2411 Micro Floppy	R3
iCOM 3712	A1
iCOM 3812	A1
iCOM 4511 Cartr. CP/M v.1.4	D1
iCOM 4511 Cartr. CP/M v.2.x	D2
IMSAI VDP-40/VDP-42	R4
IMSAI VDP-44	R5
IMSAI VDP-80	A1
Industrial Microsystems 5000	RA
Industrial Microsystems 8000	A1
Intel iPDS	M6
Intel MDS SD	A1
Intersil Development Sys	A1
Inter Systems Ithaca 800	A1
Intertec Superbrain DOS 0.5-2.x	RJ
Intertec Superbrain DOS 3.x	RK
Intertec Superbrain QD	RS
ISC Intecolor 8063/8360/8963	A1
Lanier EZ-1	M3
Lanier Super	Q4
Lexitron VT 1303 DS/DD	S8
Lexor Alphasprint Model S1	S1
Lexor Lexoriter	S1
Meca Delta-1 5¼″	P6
MICOM 2001	B3
MICOM 2001E	B4
MICOM 3003	M1
Micromation	A1
MicroMega 85	SC
Micropolis Mod 1	Q1
Micropolis Mod II	Q2
MITS 3200-3202	B1
Monroe OC 8820, DD/SS	SW
Morrow Discus	A1
Mostek	A1
MSD 5¼″	RC
MULTI-TECH-I	Q2
MULTI-TECH-II	Q2
Nascom (Gemini drives)	R3
Nascom II with Lucas Drives	SL
National MSC 6600	A1
NCR 8140/9010	A1
NEC PC-8001	RV
Nicolet Logic Analyzer Model 764	SX
NNC-80/80W	A1
North Star SD	P1
North Star DD	P2
North Star QD	P3
Northern Telecom 503	SM
Nylac Micropolis Mod II	Q2
Ohio Scientific C3	A3
OKI iF-800 + MSA CP/M-80	SP
OKI iF-800 + OKI/LB CP/M-80	SR
Osborne-1	SA
Otrona Attache	MC
Pertec PCC 2000	A1
PET/CBM + SSE Bx + 8050	C2
PET/CBM w/Madison Z-RAM + 8050	C4
Philips P-2000	MA
Philips MICOM 2001 8″	B3
Philips MICOM 2001E	B4
Philips MICOM 3003	M1
Processor Technology Helios II	B2
Quasar QDP100	A1
Quay 500	RQ
Quay 520	RP
Quay 900	A1
RAIR DD	RE
RAIR SD	R9
Research Machines 5.¼″	RH
Research Machines 8″	A1
Sanco 7000 5″	RQ
Sanyo MBC 1000	SY
Sanyo MBC 2000	SS
Sanyo MBC 3000	A1
Seattle	E1
Sony	U1
SD Systems 5¼″	R3
SD Systems 8″	A1
Spacebyte	A1
Tarbell 8″	A1
TecMar	E1
TEI 5¼″	R3
TEI 8″	A1
Televideo DD/DS	S5
T.I.P. (Alloy Engineering, Inc.)	T3
Toshiba T200	SF
Toshiba T250	A1
Triumph Adler Alphatronic	SV
TRS Model I + Omikron 5¼″	RM
TRS Model 1 + FEC Freedom	RN
TRS-80 Model 1 + Shuffleboard	A1
TRS-80 Model II	A1
Vector MZ	Q2
Vector System 2800	A1
Vector System B/VIP	Q2
Vista V-80 5¼″ SD	R8
Vista V200 5 DD	P6
Wangwriter	SE
WORDPLEX	SZ
XEROX 820, 5¼″	S6
XEROX 820, 860 8″	A1
ZEDA 580	SH
Zenith Z89 + Magnolia CP/M-80	P7
Zenith Z89 + Zenith CP/M-80	P7
Zenith DD/SS	SK
Zenith DD/DS	SJ
Zilog MC 22-20/25/50	A1

Program names and computer names are generally trademarks or service marks of the author or manufacturing company.

All Lifeboat (LB) 8-bit software requires SB-80 (or other CP/M-80 compatible disk operating system) unless otherwise stated.

All products are subject to terms and conditions of sale.

Feature SETATR CP/M File Attributes Program

Thomas N. Hill

Introduction

In the November issue I described a program which replaced the IOBYTE handling methods of the CP/M-80 program STAT with a user friendly, menu driven program called "SETIO". In this article, I will present a companion program called "SETATR" which replaces the STAT manipulations of the file attribute flags implemented in CP/M-80 2.x.

About File Attributes

When Digital Research released version 2.0 of CP/M-80, they provided the system user with a measure of file security by implementing two 'file attribute' flags. One flag, designated as $DIR/$SYS, controlled the directory display of the file(s) the flag was associated with. When the flag was SET, that file was not displayed in the disk directory. This allowed users to effectively 'hide' programs which were considered potentially dangerous when used by the inexperienced, and let them relieve the clutter in the directory of 'system' programs which appeared upon every disk used in a system. The second flag, termed the $R/O/$R/W flag, provided another measure of file use control by allowing files to be designated as 'Read Only' or 'Read Write'. By setting a file to Read/Only, the user protected that file from being written to by other users or programs.

When MP/M II was released, Digital Research gave us another flag to play with, called the 'Archive' flag. This flag, in conjunction with a new option for the PIP program, allowed selective disk to disk copying of only those files which had been altered since the last copy operation. This made possible a practical and almost automatic method of periodic file backups. Recently, I took note that Kelly Smith, a prolific writer of very useful programs, had released to the public via *Lifelines/The Software Magazine* a program titled "ARCHIVE" which took essentially the same semi-automatic file backup methods developed for MP/M II and applied them to the CP/M-80 2.x operating system.

What STAT Does with File Attributes

The CP/M program STAT implements a rather restrictive way of manipulating file attribute flags. The general STAT command to alter these flags is:

```
A>STAT <filename> $<attribute>
```

where <filename> is a valid CP/M-80 file reference, possibly containing wildcard characters, and <attribute> is one (and only one) of the following attribute designators:

R/W — Set the file(s) to Read/Write status,
R/O — Set the file(s) to Read/Only status
SYS — Set the file(s) to System (hidden) status,
DIR — Set the file(s) to Directory (visible) status

Using the STAT program, a user must set all the .COM files to R/O in two STAT commands, since only one attribute change can be placed upon a command line.

Introducing SETATR

The program SETATR was developed to provide an easier, more flexible way of altering the file attributes. The primary goal was to change more than one of the flags at any one time. It was also desirable to implement the Archive flag, for those users who are using Kelly Smith's "ARCHIVE" program or are operating under an MP/M II environment. A secondary design goal was to provide intelligent, meaningful error messages for non-technical users. With the rapidly dropping cost of memory and secondary storage, there is little excuse for terse or cryptic error messages merely as a space saving measure.

Unlike the SETIO program, SETATR is driven entirely from the CP/M-80 command line. All program commands and inputs must be placed upon the input line following the program name. The format of the command line is:

```
A>SETATR <filename> /<list of attribute codes>
```

where <filename> is a valid CP/M-80 file reference, possibly including wildcards, and the string following the "/" is a series of file attribute identifiers to change. The attribute identifiers may be in any order and may contain any combination of attributes.

If conflicting attribute sets are requested, the program informs the user of this fact and aborts the program with no file changes. The attribute identifiers allowed upon the command line are:

S — System attribute, no directory display,
D — Directory attribute, visible in directory,
R — Read Only attribute,
W — Read Write attribute,
A — Archive attribute.

The Archive attribute identifier also must be followed by either a plus (+) or a minus (−) sign, indicating the respective setting or resetting of the Archive flag. An example of a valid SETATR command line could be:

```
A>SETATR *.COM /RSA-
```

This command would set the Read/Only and System attributes and reset the Archive attribute for all the .COM files upon the default disk.

In keeping with the design goal of intelligent error messages, the following are examples of error messages produced by the program:

"Missing or invalid option list marker, must be '/' "
"Invalid character following Archive attribute option. An "A" must be followed by either a "+" or a − " to indicate setting or resetting of the Archive flag."
"Attribute flags in conflict. You may not set both the

S)ystem and D)irectory flags, nor the R)ead-only and read-W)rite attributes simultaneously."

The rest of the error messages are equally informative concerning the cause for error, and in many cases attempt to provide instruction so the error may be corrected at the next invocation of the program.

Inside the SETATR Program

The first page or two of the program listing is comprised of standard CP/M-80 system equates, derived from a library file and incorporated into the program with the MACLIB pseudo-op provided in the Digital Research MAC macro-assembler.

The SETATR program is divided into three main sections labeled INIT, PARSE, and EXECUTE, respectively. Each section's function is implied by its name. The INIT routine initializes memory flags, checks for the presence of a command tail in the CPM/M-80 command buffer, and, if a command tail is present, converts it to uppercase before passing it to the PARSE routine. In the case of a missing command tail, the appropriate error message is displayed and the program terminates.

The PARSE routine examines the command tail for proper syntax and prepares the various flag indicators needed by the EXECUTE routine. The first argument upon the command tail (i.e., up to the first space) is parsed as a CP/M-80 file reference into the default File Control Block. Then a scan is made over the line in search for a slash (/), which functions as the attribute option list marker. If no slash is found in the line, again an appropriate error message is output and the program terminates. Once the slash is found, each character following it is examined for validity against the various attribute tokens (S,D,R,W,A).

When one of a pair of attributes is parsed, the rest of the command line is scanned for an occurrence of the opposite attribute token. If one is found, an error message describing conflicting attributes is displayed and the program terminates. Also checked is the presence of one of the two Archive attribute modifiers. If neither is present following an "A" token, the program displays an error message and terminates. If the parser reaches the end of the command line without aborting due to syntax errors, it returns control to the main program loop, which passes control to the EXECUTE routine.

The EXECUTE routine uses the flags set and/or reset by the PARSE routine to determine which flag bits of the selected file(s) to alter. The flags are in one of three states, depending upon 1) the presence or absence of the attribute, and 2) if present, the selected state of the attribute. If the attribute was not included in the command tail option list, the flag will be set to 0FFH. If the attribute is to be reset, (i.e., Directory, Read-Wrte, or Archive OFF), then the flag will be 00 H. If the attribute is to be set then the flag will be set to 80H. The EXECUTE routine uses the parsed File Control Block to gather the selected files into a memory table, which is then used to build new File Control Blocks with the proper attribute flags appropriately set or reset. This File Control Block is then used as the argument to the CP/M-80 BDOS function 30, (Set Attribute), to record the new attribute information in the disk directory.

Several auxiliary subroutines are used by the three main routines, among them are routines which gather together files matching an ambiguous file reference, display the current file and describe which attributes are being altered, and other small routines which convert lower to upper case, print messages, and scan strings for specified characters.

Summary

The SETATR program described here, and its companion SETIO, correct some observed deficiencies in the CP/M-80 STAT program. By providing flexible, friendly programs to deal with the alterations of system and file parameters, I have attempted to bridge the all too present gap between the non-technical computer user and the operating system/ computer interface.

Like the SETIO program before it, the SETATR program was developed to encourage office personnel to use the computer system by providing a friendlier environment. By providing informative error messages and a flexible command syntax, I feel that they succeed. The SETATR program has been in operation for some four months now, and most of the bugs have been discovered and DDTed. Feel free to drop me a line if you discover any that I have missed.

```
TITLE 'SETATR   CP/M FILE ATTRIBUTES PROGRAM'

; This program and SETIO mark the death of
; the CP/M utility STAT. SETIO gives the user
; full control, via menu-driven utility,
; of the IOBYTE used by CP/M.
; This program provides a similar function
; for modification of the file attributes
; $SYS,$DIR,$RO, and $RW.
; Extensions to STAT include provisions
; for multiple attribute controls
; on files (with automatic check for
; conflicting sets), and control of
; the Archive attribute introduced
; with MP/M II.
; WRITTEN BY:  Thomas N. Hill
;              Alaska Micro Systems
;              200 Oklahoma St.
;              Anchorage, Ak.   99504
;              (907) 337-1984  (9 AM - 5 PM, AST)
;
; Modification and Update list:
; Version 1.0 Implemented, June 21, 1982 (TNH)
; SYSTEM EQUATES

        CPM      EQU     0
        BDOS     EQU     CPM+5H  ; BDOS ENTRY PT
        FCB1     EQU     CPM+5CH ; CP/M FILE CTRL BLOCK
        FCB2     EQU     CPM+7CH ; 2ND FILE CTRL BLOCK
        CBUF     EQU     CPM+80H ; DEFAULT COM BUFFER
        TPA      EQU    CPM+100H;USRPROG AREA

; NON-DISK I/O FUNCTIONS

        CONIN    EQU      1     ; CONSOLE INPUT
        CONOUT   EQU      2     ; CONSOLE OUTPUT
        LSTOUT   EQU      5     ; LIST DEVICE OUTPUT
        PRTBUF   EQU      9     ; SEND STRING TO CONSOLE
        RDBUF    EQU      10    ; GET STRING FR/CONSOLE
        CONSTAT  EQU      11    ; CONSOLE STATUS
        VERS     EQU      12    ; RET CP/M (MP/M) VERS #

; DISK I/O FUNCTIONS
```

(continued on next page)

```
        SELDSK  EQU     14      ; SELECT DISK
        OPENF   EQU     15      ; OPEN FILE
        CLOSEF  EQU     16      ; CLOSE A FILE
        DELETF  EQU     19      ; DELETE A FILE
        READF   EQU     20      ; READ A RECORD
        WRITEF  EQU     21      ; WRITE A RECORD
        MAKEF   EQU     22      ; CREATE A FILE
        SETDMA  EQU     26      ; SET DISK DMA ADDRESS
        SIZEF   EQU     35      ; COMPUTE FILE SIZE
        SERCHF  EQU     17
        SERCHN  EQU     18
        ATTSET  EQU     30
        GETDRV  EQU     25      ; RET CURRENT DISK

; THOSE FUNCTIONS REQUIRING A BYTE
; ARGUMENT WILL EXPECT THAT BYTE
; TO BE IN THE E REGISTER.  ADDRESS
; ARGUMENTS ARE PASSED IN THE
; DE REGISTER.  RETURN CODES ARE
; PASSED IN THE ACC.  IN GENERAL,
; A RETURN OF ZERO INDICATES SUCCESS,
; WHILE A OFFH INDICATES FAILURE.

; CHARACTER EQUATES

        CR      EQU     0DH; CARRIAGE RETURN
        LF      EQU     0AH; LINE FEED
        ESC     EQU     1BH; ESCAPE CODE
        EOF     EQU     1AH; END-OF-FILE, CTRL Z
        BELL    EQU     07H; TERMINAL BELL
        BS      EQU     08H; BACKSPACE
        TAB     EQU     09H; TAB CHAR
        ;
        FALSE   EQU     00H
        TRUE    EQU     0FFH
        ;

        ORG     TPA

SETATR: CALL    INIT            ; SET THINGS UP
MAIN1:  CALL    PARSE           ; PARSE INPUT LINE
        JNZ     FIN             ; ERROR IN COM LINE
        CALL    EXECUTE         ; PERFORM SETS DONE
FIN:    JMP     CPM

; SUBROUTINES BEGIN HERE
; HERE WE INITIALIZE AND CHECK FOR COM INPUT LINE.

INIT:   LDA     CBUF
        ORA     A               ; ANYTHING IN COM LINE?
        RZ
        MOV     B,A             ; SAVE COUNT
        LXI     H,CBUF+1
        LXI     D,CBUF+2        ; SET UP TO CONVERT TO UCASE
INIT1:  LDAX    D               ; AND IGNORE LEADING SPACE
        CALL    UCASE
        MOV     M,A
        INX     H
        INX     D
        DCR     B
        JNZ     INIT1           ; COM LINE NOW UCASE
        RET

; PARSE ROUTINE TO ACCEPT UCASE COM LINE
; AND USE 1ST ARGUMENT AS FILE REFERENCE. THE REST OF
; THE LINE IS PARSED FOR FILE ATTRIBUTE INFO.

PARSE:  LXI     H,CBUF+1
        MVI     B,´/´           ; 1ST CHECK FOR RIGHT OPTION
        PUSH    H               ; MARKER
        CALL    SCAN
        POP     H               ; RESTORE POINTER
        JZ      PARSE1
        LXI     D,NOMARK
        JMP     PERROR          ; NO VALID MARKER
PARSE1: LXI     D,FCB1
        CALL    SETFILE         ; SET UP FCB FR/COM LINE
        MVI     B,´/´
        CALL    SCAN            ;DO REAL SCAN TO
        INX     H               ; OPTION LIST
PARSE2: MOV     A,M             ; GET 1ST OPTION
        CPI     ´S´             ; SET TO SYSTEM?
```

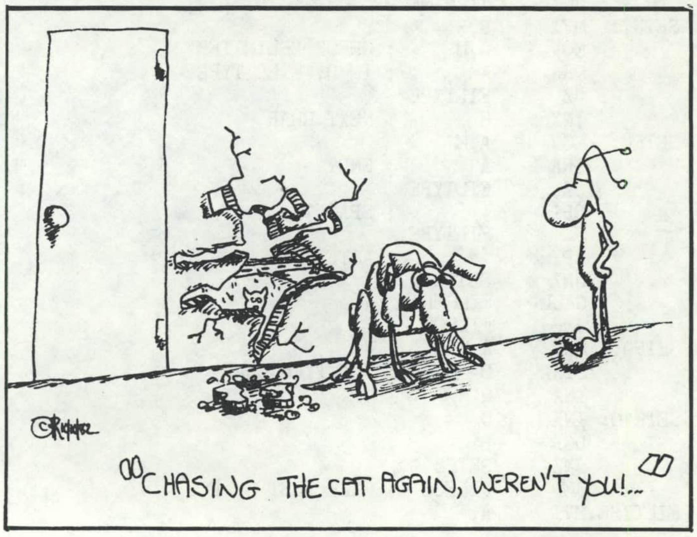
```
        JZ      GOTSYS          ; HAVE A SYSTEM FLAG
        CPI     ´D´
        JZ      GOTDIR          ; GOT DIRECTORY FLAG
        CPI     ´R´
        JZ      GOTRO           ; GOT READ ONLY FLAG
        CPI     ´W´
        JZ      GOTRW           ; GOT READ-WRITE FLAG
        CPI     ´A´
        JZ      ARCHIVE         ; GOT ARCHIVE FLAG
        CPI     0               ; END OF STRING
        RZ
        LXI     D,BADFLAG
        JMP     PERROR          ; AND ABORT

; INDIVIDUAL FLAG ROUTINES
; SYSGEM FLAG SET HERE, CHECK FOR DIR IN LINE
; ABORT AS ERROR IF D IS FOUND

GOTSYS: INX     H               ; NEXT CHAR POSITION
        PUSH    H
        MVI     B,´D´
        CALL    SCAN
        POP     H               ; SCAN RETURNS 0 SET
        JZ      CNFERR          ; IF FOUND
GOTS1:  MVI     A,80H
        STA     SDFLG           ; SET SYSTEM FLAG
        JMP     PARSE2          ; CONTINUE

        ; DIRECTORY FLAG HERE

GOTDIR: INX     H
        PUSH    H
        MVI     B,´S´           ; LOOK FOR CONFLICT
        CALL    SCAN
        POP     H
        JZ      CNFERR          ; CONFLICT ERROR
        MVI     A,0
        STA     SDFLG           ; SET DIRECTORY FLAG
        JMP     PARSE2

; READ-ONLY FLAG HERE

GOTRO:  INX     H
        PUSH    H
        MVI     B,´W´           ; CONFLICT CHECK
        CALL    SCAN
        POP     H
        JZ      CNFERR
        MVI     A,80H
        STA     RFLG            ; SET READ-ONLY FLAG
        JMP     PARSE2

; READ WRITE FLAG HERE

GOTRW:  INX     H
        PUSH    H
        MVI     B,´R´           ; CONFLICT CHECK
        CALL    SCAN
        POP     H
        JZ      CNFERR
        MVI     A,0
        STA     RFLG            ; RESET READ/WRITE FLAG
        JMP     PARSE2

; ARCHIVE FLAG HERE, NO CONFLICT CHECKS NEEDED.
; CHECK NEXT CHAR FOR STATE TO SET FLAG,
; "-" TO RESET, "+" TO SET.

ARCHIVE:
        INX     H
        MOV     A,M
        INX     H
        CPI     ´-´             ; RESET?
        JZ      ARCHOFF
        CPI     ´+´
        JZ      ARCHON          ; OR SET?
        LXI     D,BADARCH       ; NEITHER,STATE ERROR
        JMP     PERROR          ; RETURN ERROR TO MAIN LOOP
ARCHON: MVI     A,80H
        STA     ARCFLG          ; TURN ON ARCHIVE FLAG
        JMP     PARSE2
ARCHOFF:
        MVI     A,0
        STA     ARCFLG    ; OR TURN IT OFF
        JMP     PARSE2

; CONFLICT ERROR,GENERAL MESSAGE AND ABORT

CNFERR: LXI     D,CONFLICT

; PARSE ERROR, PRINT STRING AND RETURN
; ERROR CODE TO MAIN LOOP

PERROR: CALL    PSTRING
        XRA     A
        INR     A          ; RESET ZERO FLAG
        RET

; FILE CONTROL BLOCK BUILDER

SETFILE:
        INX     H        ; POINT TO 2ND CHAR
        MOV     A,M      ; OF STRING
        CPI     ':'      ; IS COLON?
        MVI     A,0
        DCX     H
        JNZ     SETF1    ; NO, USE DEFAULT DRIVE
        MOV     A,M      ; CHANGE DRVES,GET NAME
        SUI     40H      ; MAKE TRUE NUMBER
        PUSH    H
        PUSH    PSW
        MVI     C,GETDRV;FIND OUT WHERE WE ARE NOW
        CALL    BDOS
        STA     CDRIVE   ; SAVE AS CURRENT DRIVE
        POP     PSW
        POP     H
        INX     H
        INX     H        ;PT TO 1ST CHAR OFFILE NAME
SETF1:  STAX    D        ; SET DRIVE FIELD
        INX     D
        MVI     B,8      ; CHARS IN NAME
SETF2:  MOV     A,M
        ORA     A        ; IS 0? (END OF STRING)
        JZ      FILNAME  ; FILL NAME WITH SPACES
        CPI     '.'      ; END OF NAME?
        JZ      FILNAME  ; FILL WITH BLANKS
        CPI     '*'      ; IS ASTERISK?
        JNZ     SETF3
        CALL    FILQMRK  ; FILL WITH QMARKS
        JMP     SETF7    ; FILL WITH QMARKS
SETF3:  MOV     A,M      ; REGET CHARACTER
        STAX    D        ; PUT IT AWAY
        INX     H
SETF4:  INX     D        ; ADVANCE POINTERS
        DCR     B
        JNZ     SETF2    ; NEXT CHARACTER
SET5:   JMP     SETF7    ; DO TYPE
FILNAME:MVI     A,' '
        STAX    D
        INX     D        ; FILL REST OF NAME W/SPACE
        DCR     B
        JNZ     FILNAME
SETF7:  MVI     B,3
        MOV     A,M      ; CHECK DELIMITER
        ORA     A        ; IF 0, FILL TYPE
        JZ      FILTYPE
        INX     H        ; NEXT CHAR
SETF8:  MOV     A,M
        ORA     A        ; END?
        JZ      FILTYPE
        CPI     ' '      ; SPACE?
        JZ      FILTYPE
        CPI     '*'      ; ASTERISK?
        JNZ     SETF9
        CALL    FILQMRK
        JMP     FILFCB
SETF9:  MOV     A,M
        STAX    D        ; SET TYPE CHAR
        INX     H
SETF10: INX     D
        DCR     B
        JNZ     SETF8
        JMP     FILFCB   ; FILE REST OF FCB
FILTYPE:MVI     A,' '
```

```
        STAX    D
        INX     D
        DCR     B        ; FILL REST OF TYPE W/SPACE
        JNZ     FILTYPE
FILFCB: MVI     B,24
        MVI     A,0      ; FILL REST OF FCB W/0'S
FILFCB1:STAX    D
        INX     D
        DCR     B
        JNZ     FILFCB1
        RET

FILQMRK:MVI     A,'?'
FILQ:   STAX    D
        INX     D
        DCR     B
        JNZ     FILQ
        INX     H
        RET

; HERE WE SET (OR RESET) THE REQUESTED ATTRIBUTES.

EXECUTE:CALL    GETFILES ; ACCUMULATE FILES
EXEC0:  LHLD    FILPTR   ; GET POINTER TO FILES
        MOV     A,M
        INR     A        ; END OF LIST?
        JZ      CPM      ; DONE.
        LXI     D,TSTFCB+1 ; MOVE FILE TO WORK FCB
        INX     H        ; PAST USER NUMBER
        LDA     FCB1     ; GET DRIVE CODE
        STA     TSTFCB
        MVI     B,11
EXEC1:  MOV     A,M
        STAX    D        ; MOVE NAME AND TYPE
        INX     H
        INX     D
        DCR     B
        JNZ     EXEC1
        SHLD    FILPTR
        MVI     A,0
        STAX    D        ; MARK END
        LXI     H,TSTFCB+1
        CALL    FILEOUT

; ACTUAL FILE NAME IN THE WORK FCB, ALTER ATTRIBUTES
; AND SET THEM

        LXI     H,TSTFCB+9; GET TO R/O ATTRIBUTE
        MOV     B,M
        LDA     RFLG
        CPI     TRUE     ; SKIP THIS FLAG?
        JZ      EXEC2+1
        ORA     A        ; IS IT TRUE OR FALSE
        JZ      RESRO    ; FALSE, RESET IT
        ORA     B        ; ELSE TRUE, SET IT.
        JMP     EXEC2
RESRO:  MOV     A,B
        ANI     7FH      ; RESET FLAG
EXEC2:  MOV     M,A      ; PUT CHAR BACK
        INX     H
        MOV     B,M      ; SYSTEM ATTR
        LDA     SDFLG
        CPI     TRUE
        JZ      EXEC3+1
        ORA     A
        JZ      RESDIR   ; SET TO DIR
        ORA     B        ; ELSE SET SYS ATTR
        JMP     EXEC3
RESDIR: MOV     A,B
        ANI     7FH      ; RESET TO DIR
EXEC3:  MOV     M,A
        INX     H        ; ARCHIVE FLAG BIT
        MOV     B,M
        LDA     ARCFLG
        CPI     TRUE
        JZ      EXEC4+1
        ORA     A
        JZ      RESARC
        ORA     B        ; SET ARCHIVE BIT
        JMP     EXEC4
RESARC: MOV     A,B
        ANI     7FH      ; RESET ARCHIVE BIT
```

(continued on next page)

```
EXEC4:  MOV     M,A
        LXI     D,TSTFCB
        MVI     C,ATTSET; SET ATTRIBUTES
        CALL    BDOS

; CONTINUE UNTIL SEARCH FOR FILES FAILS.

        JMP     EXEC0   ; CONTINUE


; GATHER UP ALL MATCHING FILES

GETFILES:
        LXI     H,FILEBUF
        SHLD    FILPTR  ; SET UP POINTER
        LXI     D,DIRBUF
        MVI     C,SETDMA
        CALL    BDOS    ; SET UP DIR BUFFER
        LXI     D,FCB1
        MVI     C,SERCHF; LOOK FOR 1ST MATCH
        CALL    BDOS
        INR     A       ; ANYTHING THERE?
        JZ      NOFILES ; NO
GETF0:  CALL    INDEX   ; FIND OUT WHERE IT IS
        XCHG
        LHLD    FILPTR  ; MOVE FILE NAME TO BUFFER
        MVI     B,12    ; CHARS TO MOVE
GETF1:  LDAX    D
        MOV     M,A
        INX     H
        INX     D
        DCR     B
        JNZ     GETF1
        SHLD    FILPTR  ; SAVE NEW POINTER
        LXI     D,FCB1
        MVI     C,SERCHN; GET THE NEXT ONE
        CALL    BDOS
        INR     A       ; ANY LEFT?
        JNZ     GETF0   ; YEP, MOVE IT.
        LHLD    FILPTR  ; NO, MARK END OF LIST
        MVI     M,0FFH
        LXI     H,FILEBUF ;RESET POINTER TO
        SHLD    FILPTR  ; START OF BUFFER
        RET

FILEOUT:
        MOV     A,M     ; PRINT STRING AT (HL) TIL 0
        ORA     A
        JZ      FILE01
        MOV     E,A
        PUSH    H
        MVI     C,CONOUT
        CALL    BDOS
        POP     H
        INX     H
        JMP     FILEOUT
FILE01: LXI     D,SETTO
        CALL    PSTRING ; "SET TO"
        LDA     RFLG
        CPI     TRUE
        JZ      LBL1
        LXI     D,ROMSG
        ORA     A       ; IS R/W?
        JNZ     FILE02
        LXI     D,RWMSG
FILE02: CALL    PSTRING
LBL1:   LXI     D,SYSMSG
        LDA     SDFLG
        CPI     TRUE
        JZ      LBL2
        ORA     A
        JNZ     FILE03
        LXI     D,DIRMSG
FILE03: CALL    PSTRING
LBL2:   LXI     D,ARMSG1; ARCHIVE ON?
        LDA     ARCFLG
        CPI     TRUE
        JZ      LBL3
        ORA     A
        JNZ     FILE04
        LXI     D,ARMSG2
```

```
FILE04: CALL    PSTRING
LBL3:   CALL    CRLF
        RET

; CALCULATE INDEX TO FILE DIR ENTRY.

INDEX:  DCR     A; TAKE CARE OF ERROR TEST
        ADD     A
        ADD     A; (A) TIMES 5
        ADD     A
        ADD     A
        ADD     A
        MOV     E,A
        MVI     D,0
        LXI     H,DIRBUF ; SET UP FOR INDEX
        DAD     D        ; DO IT.
        RET

; NO FILES FOUND WHICH MATCH REQUESTED REFERENCE.

NOFILES:
        LXI     D,NOFLMSG
        CALL    PSTRING
        JMP     CPM      ; EXIT TO CP/M.

; CONVERT THE CHAR IN THE ACC TO UCASE

UCASE:  CPI     ´a´
        RC
        CPI     ´z´+1
        RNC
        ANI     5FH
        RET

; PRINT STRING POINTED TO BY DE.

PSTRING:MVI     C,PRTBUF
        JMP     BDOS

; SEND CR, LF TO CONSOLE

CRLF:   LXI     D,CRLFMSG
        JMP     PSTRING

; SCAN0 BUFFER AT (HL) FOR THE CHARACTER IN (B).
; STOP ON MATCH OF 0 BYTE

SCAN:   MOV     A,M
        ORA     A       ; ZERO?
        JZ      NOFIND
        CMP     B       ; CHAR MATCH?
        RZ
        INX     H
        JMP     SCAN
NOFIND: INR     A       ; RESET 0 FLAG
        RET

; DATA AREAS
; MESSAGES
CRLFMSG:DB      CR,LF,´$´
```



```
NOMARK: DB       bell,'Missing or invalid option list marker, must be "/"'
        DB       cr,lf,'$'
BADFLAG:DB       bell,'Ill-formed or illegal file attribute. Option list must'
        DB       cr,lf,'consist of one or more of the following:',cr,lf
        DB       tab,'S)ystem',cr,lf
        DB       tab,'D)irectory',cr,lf
        DB       tab,'R)ead-only',cr,lf
        DB       tab,'read-W)rite',cr,lf
        DB       tab,'A)rchive',cr,lf
        DB       tab,'(NOTE: An "A" MUST be followed by either a "+" or a "-"'
        DB       CR,LF
        DB       tab,'      to indicate setting or resetting the flag.)',cr,lf
        DB       '$'
BADARCH:DB       bell,'Invalid character following A)rchive attribute option.'
        DB       CR,LF
        DB       tab,'(NOTE: An "A" MUST be followed by either a "+" or a "-"'
        DB       CR,LF
        DB       tab,'      to indicate setting or resetting the flag.)',cr,lf
        DB       '$'
CONFLICT:
        DB       bell,'Attribute flags ar in conflict.  You may not set both'
        DB       'the S)ystem and D)irectory flags, '
        DB       'nor the R)ead-only and read-W)rite attributes '
        DB       'simultaneously.',cr,lf,'$'
SETTO:  DB       ' set to $'
ARMSG1: DB       ' Archive ON$'
ARMSG2: DB       ' Archive OFF$'
SYSMSG: DB       ' SYSTEM,$'
DIRMSG: DB       ' DIRECTORY,$'
ROMSG:  DB       'READ-ONLY,$'
RWMSG:  DB       'READ-WRITE,$'
NOFLMSG:DB       'No files found which match the supplied reference.',cr,lf,'$'

        ; flag storage

SDFLG:  DB       0FFH
RFLG:   DB       0FFH
ARCFLG: DB       0FFH
CDRIVE: DB       0
FILPTR: DW       FILEBUF

        ; file control blocks.

TSTFCB: DB       0,'           '
        DW       0,0,0,0,0,0,0,0,0,0,0,0

DIRBUF: DS       80H

FILEBUF:DS       1

        END
```

New Versions

- BSTAM-86 for the IBM-PC ver. 4.6 available for CP/M-86. Note: BSTAM-86 will run under MS-DOS if the customer has the EMULATOR-86.
- Dental Management System (Univair's Series 9000) now ver. 2.06
- Medical Management System (Univair's Series 9000) now ver. 2.06
- dUTIL (new product) ver. 1.1 — for dBASE-II
- QUICKCODE (new product) ver. 2.1 — for dBASE-II

THOUSANDS OF CP/M SOFTWARE PACKAGES ARE GATHERING IN SAN FRANCISCO

CP/M '83®

Sponsored by
DIGITAL RESEARCH™
The Creators of CP/M

Moscone Center, San Francisco
Friday-Sunday, January 21-23, 1983
Show hours: **11 AM to 6 PM Daily**
Seminars (for the trade): 9 AM to 11 AM
User Conferences and Workshops: 11 AM to 5 PM

Beginning January 21 your CP/M computer will never be the same.

On January 21, you are going to discover an incredible world of software, services and support for your CP/M computer. January 21 marks the beginning of CP/M '83 . . . a 3-day-ong international extravaganza for CP/M users. For the first time ever, you'll be able to learn about, try out and compare iterally thousands of products for your system. All of the CP/M support companies . . . large and small . . . will be there, demonstrating the newest, most innovative CP/M products. Leading authorities from the software industry will run dozens of in-depth seminars and workshops to help you explore the full potential of your CP/M computer.

CP/M '83 will actually be the *largest first-year computer show in history*! More than 25,000 CP/M users from around the world will view nearly 700 exhibits and attend more than 100 workshops.

he World Under One Roof

At CP/M '83, you will find everything new for your CP/M computer under one roof. In a couple of days, you can sample software, accessories and services for every conceivable application you may have. You can talk to experts and learn about both the features *and* limitations of these products. You can 'test drive' software and purchase it right at the show . . . at special show prices. You can attend seminars, workshops and panel discussions, presented by the most renowned authorities in the software field, and learn how to make your computer more productive and more powerful. You'd have to spend months visiting computer stores and reading trade journals before you'd find even a fraction of the information you'll get at CP/M '83. CP/M '83 will bring it all together . . . under one roof.

The World's Leading CP/M Experts

CP/M '83's Conference and Seminar program will include noted leaders from the industry including Gary Kildall, President, Digital Research Inc.; Sol Libes, Editor, Micro Systems Magazine; Christopher Morgan, Editor-in-Chief, Byte and Popular Computing Magazines; Adam Osborne, President, Osborne Computer Corporation; Tony Gold, Founder CP/M Users Group and Lifeboat Associates; Ben Rosen, President, Rosen Research; Portia Isaacson, President, Future Computing; Maggie Cannon, Editor in Chief, InfoWorld; David Crockett, Senior Vice-President, Dataquest; and Gordon Eubanks, Vice-President Language Div., Digital Research. CP/M '83's conference, seminar and workshop program is designed for computer tradespeople and users. These individuals, plus dozens of others, will conduct informative discussions exploring CP/M applications, technical information, development aids, venture capital programs and software distribution. End user workshops will show users how to get the most from their CP/M computer.

how & Conference Preregistration Request

. Complete this form (or a facsimile) and mail it with a check to CP/M '83 to The National Computer Shows, 822 Boylston Street, Chestnut Hill, MA 02167.

. All preregistration requests must be received by Friday, January 14, 1983. No telephone or credit card orders can be processed.

. Use a separate form for each person preregistering for a three day badge.

4. Badges and tickets will be mailed back to preregistrants, providing the order is received by Friday, January 7. For orders received after that date, badges and tickets will be held for pick-up at the preregistration desk at the Show.

5. It is recommended that attendees preregister. However, it is not necessary, as badges and tickets can be purchased at the Show.

6. Persons preregistered by January 7 receive an exhibits Conference & Seminar Schedule by return mail.

ame ______________________________

ompany (if any) ______________________________

ity ____________ State ________ Zip ________

elephone (Area Code) ______________________________

Check Applicable Box:

□ Enclosed is my payment for ________ (quantity) one day, exhibits-only tickets at $10 each.

□ Enclosed is my payment of $20 for a three day exhibits and conference ticket/badge. (Use duplicate copy to order more than one).

Low Hotel Rates
CP/M '83's Show Management has made special group discount arrangements with several of the hotels closest to Moscone Center. The unbelievably low discount rates are available at such luxurious facilities as the Hyatt on Union Square, and at modest priced accommodations, including the Pickwick Hotel and Holiday Inns.

The Hyatt on Union Square—415-398-1234—345 Stockton St.—new, modern, and four blocks from Moscone. CP/M '83's discount rates are $75 per night single, $85 per night double, versus their regular rates of $95 to $130 single and $130 to $165 double.

The Sheraton Palace Hotel—415-392-8600—639 Market St.—elegantly refurbished fine older hotel, and the closest hotel to Moscone—only one block. CP/M '83's special rates are $70 single and $80 double, versus their regular rates of approximately $90 single and $100 double.

The San Francisco Hilton & Tower—415-771-1400—333 O'Farrell St.—a modern facility located three blocks from the Show. CP/M '83's special rates are $75 per night single and $95 double, versus this Hilton's regular rates of $66 to $127 single, and $86 to $147 double.

The Holiday Inn on Union Square—415-398-8900—480 Sutter St.—a modern facility with all the conveniences, and located in the heart of San Francisco. This facility offers CP/M '83 attendees accommodations for $70 per night single and $80 per night double.

The Holiday Inn Financial District/China Town—415-433-6600—750 Kearny St.—this high rise Inn is six easy blocks to Moscone and offers CP/M '83's attendees single or double accommodation at $55 per night versus their regular rates of $78 to $92.

The Pickwick Hotel—415-421-7500—85 5th St.—Just two blocks from the Show, this fine hotel has clean modest accommodations, and is offering CP/M '83's attendees rates which are lower than motel rates. The rates are $30 per night single and $35 per night double, and can't be beat anywhere in town.

Make your reservations today. Over 20,000 are expected to attend CP/M '83.

Hotel Reservation Request

How to Make Your Hotel Reservations

1. For you to receive CP/M '83's special discount convention rates, all reservations must be made on this form, or a facsimile. The form must be completed in detail, including date and hour of arrival, date of departure, and names and addresses of all persons who will occupy the room. Reservations can not be processed without this information.

2. Indicate at least three choices of hotels and rates. Requested rates cannot be guaranteed, but the Housing Bureau will make every attempt to assign rooms as near as possible to the requested rate.

3. The Housing Bureau requires written reservations. Only late requests, after Friday, January 7, will be accepted by telephone. After Friday, January 14 try to make your reservations directly with the hotel, but be advised that the hotels are expected to be full by that time. The Housing Bureau telephone number is 415-626-5500.

4. Mail this hotel reservation request directly to CP/M '83 Housing Bureau, P.O. Box 5612, San Francisco, CA 94101, and *not* to National Computer Shows.

5. Confirmations will be sent from the Housing Bureau up to two weeks prior to the event. Allow up to two weeks for processing.

6. Cancellations. Notify the CP/M '83 Housing Bureau of all cancellations up to Friday, January 7. After January 7, make cancellations directly with the hotel.
Changes. All other changes, such as arrival or departure times or changes in type of accommodations required, should be made directly with the hotels at all times.

7. Hotels will hold reservations only until 6 pm unless otherwise requested. If you are delayed in transit, phone ahead and advise the hotel of your arrival time. Reservations can be guaranteed to assure a room regardless of arrival time. However, if you do not pick up or cancel the reservation, you will be billed for one night's room rate. If you make a reservation, even a guaranteed reservation, it will be held only for that night. Thus, if you designate a Monday arrival and do not arrive until Tuesday, you will not have a room unless you notify the hotel beforehand.

Hotel Reservation Request

A. Please make the following hotel reservations
Hotel Choice
1st________________________________
2nd________________________________
3rd________________________________

B. Please enter my reservation at the hotel for
______ single room(s) at $________/ day
______ double room(s) at $________/ day
______ twin room(s) at $________/ day
______ one bedroom suite(s) at $________/ day
______ two bedroom suite(s) at $________/ day

C. Arrival date________ time________ □ am □ pm
Departure date________ time________ □ am □ pm
Occupant________________________________
Share With________________________________

D. Mail my confirmation to
Name________________________________
Firm (if any)________________________________
Address________________________________
City______________ State__________ Zip________
Telephone (Area Code)________________________________

E. Complete this form and mail to CP/M '83 Housing Bureau, P.O. Box 5612, San Francisco, CA 94101

(continued on next page)

MicroMoneymaker's Forum
$$$$$$$$$$$$Digital Dollars Department

Charles E. Sherman

CP/M-80 WORD PROCESSORS
EVALUATION CRITERIA
PALANTIR AND MAGIC WAND: THE CRITIC'S CHOICE

Last month we explored some of the shortcomings of the old workhorse, WordStar, and reviewed Quickey for those who would rather fix it up than abandon it. This month's column reviews my own choice for two outstanding CP/M-80 wordprocessing programs: Palantir and Magic Wand (aka PeachText). Why would anyone buy a dedicated system when they can have a micro with one of these for a fraction of the cost?

At the time of this writing, I know of fourteen major CP/M-80 wordprocessors. These are:

BENCHMARK, Metasoft Corporation, Casa Grande, AR
FINAL WORD, Mark of the Unicorn, Arlington, MA
MAGIC WAND (PeachText), Peachtree, Atlanta, GA
MAGIC TYPEWRITER, Cal. Dgtl. Engineering, Hollywood, CA
METATYPE, Amanuensis
PALANTIR, Designer Software, Houston, TX
PERFECTWRITER, Perfect Software Co., Berkeley, CA
SELECT, Select Info. Sys., Kensington, CA
SPELLBINDER, Lexisoft, Davis, CA
SUPERWRITER, Sorcim, San Jose, CA
TYPEMASTER, Steno-Tek Systems, Birmingham, MI
WORD RIGHT, Structured Systems Group, Oakland, CA
WORDSTAR, Micro Pro, San Rafael, CA
WRITE, Ashton-Tate, Culver City, CA

If I have left out anyone's favorite, please write and let me in on it. Of the ones on my list, Superwriter, Write and Word Right are still in Beta Test; I will be participating in testing two of them. Metatype has only just been released. I haven't seen Final Word yet, as they have only just released their review copies.

I have all the rest, and have reviewed most of them. I have not yet had time to give Benchmark a fair test, but it looks like it may turn out to be a good one. I have gone through the manuals for Magic Typewriter and Typemaster, and have had the latter up and running briefly. Both have circulated directly to the bottom of my pile of priorities and will stay there for the foreseeable future. This rejection on first impressions is well informed, but not objective or thorough.

To sum it up, I have looked carefully at Magic Wand, Palantir, Perfect Writer, Select, Spellbinder, and WordStar, and briefly at Benchmark. The only two I feel like recommending enthusiastically at this time are Magic Wand and Palantir, leaving Benchmark undeclared until I can spend more time with it. My favorites get 8 out of 10, as there is still room for improvement. Anyone wanna collaborate?

Evaluation Criteria

People can get attached to the program they are using, and may prefer the one they are used to even over another superior one. The truth is that preferences in wordprocessors can be very personal, and any of several good wordprocessors will *more or less* do. However, "more or less" isn't good enough when you are called upon as a friend or professional to recommend a program for someone else, so some effort should be made to sort out the objective considerations. I'm not trying to shove my choices down anyone's throat, but rather to show the standards I have developed for the selection process. Use them as a springboard toward your own conclusions.

For any given installation, you should start out by taking stock of the known and possible future uses for the wordprocessor. How many people will use it? What are their capabilities? Will it be used for: original composition? production typing? editing? What kind and what length of documents will be processed: short? long? business? financial? scientific? Is a mailing list needed or desired? Will there be mass mailings of personalized letters or forms? Is boilerplating desired for editing or printing letters or documents? And so on.

With these considerations in mind, and having seen the best features of the various CP/M-80 wordprocessors, I have developed some standards for selection which, very briefly, are:

1. The wordprocessor should be easy to learn and easy to use. I most definitely do not mean that it should be menu-driven since most menus impose rigidity and mostly just drive a competent user nuts. Instead, the way the program works must be logical, and preferably it should be obvious. There are very practical reasons for this, and it is not merely for the benefit of the first-time user. A program which is easy to use has a more positive effect on typing productivity and writing quality. Furthermore, whenever you want to call in a temporary typist to cover overload, illness, vacations, or for special projects, you don't want to have to spend hours or days giving training. For writers, use of the Edit features should become transparent (subconscious) fairly quickly. Nothing should come between the writer (typer) and the creative outpouring of ideas, especially not the tool one pours through. You don't want the unnecessary distraction of having to look up commands, figure out sequences, or stroke numerous keys in patterns which challenge your dexterity. The program should make optimum use of the features of any terminal for which it is fitted. Some programs do not use function keys at all, even where they are available, and others make poor use of them.

2. The wordprocessor should drive popular printers to their fullest capabilities. In business and personal affairs, good looking correspondence and documents really can matter. An idea should be given its best chance to connect. Why do you think we don't just mimeograph *Lifelines*? With a good word processor you can routinely put out material that looks really sharp, and that's worth a lot. Programs that do not support true proportional spacing of characters on letter-quality printers are behind the times. Even the output of the little Epson can be greatly improved if you make full use of its built-in features, but these are for the most part unsupported by most programs.

3. Boilerplates and Includes: Fast, flexible displays and Includes from external files are important to almost all writing, editing, and business applications. Electronic cut and paste between files has high priority on my list. Boilerplate capabilities are also extremely important. You should be able to call up coded "canned"material during either edit or print operations.

4. Business applications may require the manipulation of columns of figures and lists. For these applications, you want decimal tabbing and columnar block operations. Trying to set up, arrange, and move columns around can be very difficult without these features. you will also frequently want horizontal scrolling for documents exceeding 80 characters. Printers are made 132 columns wide for a reason, and your display should be able to match it for all the same reasons.

5. Mail lists and mass mailings of personalized documents: Almost everyone would like to maintain some sort of mail list, but some users will require real sophistication here. In order to make mass mailings of personalized form letters and documents, the program must be able to set up a document with conditional and variable statements which draw data from a mail list data file at print time. Most programs don't do much mail list management, so the program must at least be compatible with a good mail list accessory.

6. Short document production is more efficient with integrated programs, i.e., where printing and editing are done in the same program mode. For a work load with lots of small letters or memos, it takes too long to set up a formatted file, leave edit, call up print, then run the file. Another very desirable print feature is the ability to create a boilerplate file of various standard and special print formats which you can just "plug in" as required.

7. The program should have a typeahead buffer to keep track of characters entered during screen rewrites and disk operations. It should also pre-empt the screen rewrite when you enter successive scroll commands so you don't have to wait. Any good program should offer this, but some popular ones do not.

Magic Wand

Preliminaries: Since PeachTree bought Magic Wand they've been trying to get people to call it PeachText. Maybe they'll succeed. It was an excellent program when they bought it, then they tinkered with it a bit. By adding a main menu that allows the user to show the directory and copy, delete, or rename files, Magic Wand has become more accessible to people who do not know CP/M-80. They showed excellent discretion by making the menu entirely optional, so it is not imposed upon those who do not need it. Their other changes did a bit of minor damage (creating a botanical phenomenon, the PeachThorn) but the worst of these can be fixed with the patches described in the software tips section in this issue, which brings us to the best change of all. SBA, the previous owners of Magic Wand, gave very poor support which could also be described as "none," while PeachTree has been quite good. They furnished the patches.

PeachTree has also added companion packages for mail list management, spellchecking, and communications. The mail program is very good, but I don't recommend the other two when you can get Modem for less from CPMUG, and when The Word is better and costs less.

The documentation was excellent and still is, although a bit imposing, being over 400 pages long. Magic Wand comes preconfigured from the dealer for the user's particular terminal and printer. Changes in equipment configuration or default settings require a dealer's installation package, and cannot be reconfigured by the end user.

Magic Wand is two separate programs, Edit (18K) and Print (22K). The menu is 10K, if you use it. Big power in a small package could be an important consideration for some situations. I personally do not like the separation of Print and Edit, preferring programs which can print or edit from the same mode, but its overall superiority leads me to use it anyway. For workloads which consist mostly of short, fast documents (letters, memos) you might give extra consideration to an integrated program like Palantir.

Editing: The edit commands are few but quite sufficient, making it extremely easy to use. It is especially nice when installed on a terminal with function keys, as it makes such good use of them. Figure 1 (see page 33) shows the Edit command structure, both for a plain keyboard and for one with function keys, using the Televideo 950 as an example. The editing features are almost completely self-evident from a close look at the function-key layout. Perhaps the only features which need explaining are 1) the line delete key which takes two strokes to start it working, a protection against accidental overzap; and 2) the search and replace feature, which, to be brief, searches RAM from the cursor forward for any specific character string (no wild cards), and, if requested, replaces it with any other specific character string: on an individual basis, n number of times, or all automatically.

Edit has two modes, command and edit. You enter the program in the command mode which displays status information about your document, your workspace, tabs, and screen width. For some inexplicable reason PeachTree removed the very useful word count, giving only a character count. The command mode is the doorway to and from the edit mode, to and from the disk drives, and to and from CP/M-80. From the command mode you can manipulate blocks (copy, erase, move, name and save), display the disk directories, set line width for the terminal display, print draft (unformatted) copies of the file in RAM, setup spooling for background printing and set tabs. In the edit mode,

(continued on next page)

you create and edit text by typing, overtyping, and using the command keys illustrated in Fig. 1. The ESCAPE key takes you into the command mode; RETURN takes you back to the text.

Boilerplate and Include functions are very efficiently conducted from the command mode. You can call up and display the contents of any external file, and you can include any screen from any file into your text at any point indicated by the cursor. You can also call in specified portions of a coded boilerplate file for inclusion in your text. This is extremely important for many business applications. Also, even without the Magic Mailer program, a very rudimentary and hodge-podge sort of a mail list can be made and used as a boilerplate file; it's not much, but better than nothing.

If you have used other wordprocessors, there may very well be some favorite command which you will miss, but on an overall evaluation, Magic Wand is hard to beat as an editor. If you like tons of commands, try Perfect Writer (145K + 64K on every data disk; over 80 commands often taking three or four keystrokes; plan on days to weeks to break in new users). Missing features which may be important to some applications are: no columnar block moves, no decimal tabbing, and no horizontal scrolling.

File handling, which is executed from the command mode, is excellent. You name your file when you invoke Edit, and you may also enter a new name for the edited product at the same time or when you finish. The program remembers the file name, so you do not have to retype it, and backup files are automatically kept. You can make a temporary save without clearing memory merely by typing X <CR> in the command mode before making a dash for the ringing phone or the potty. Barring unforeseen trauma, the cursor will be right where you left it when you get back. Disk-full errors are handled easily and without loss of data by several methods: you can look at the directory and delete some unwanted file, or you can mark and save the whole file as a block on another disk. The maximum convenient size of a working file is determined by the amount of workspace you have in RAM, about 42,000 characters or 6,100 words for a 64K system. RAM overflow is heavily defended by persistent bells and flashed warnings way ahead of time, and block moves and includes are automatically defeated if they would result in an overflow condition. You can work on larger files of any size, but this requires manual manipulation of Write and Read commands, and so is inconvenient.

Error Trapping: This is an extremely stable, safe program. It doesn't invite errors, there are no traps or pitfalls, and no dead ends. Power or equipment failure is about all that will dump you back to CP/M-80 or otherwise lose you so much as a dot.

Printing: During edit, the document can be fitted with embedded commands for printing, or defaults can be relied upon, or commands can be entered at print time. These commands give you full control of all features of the letter-quality printers. They do not do as well for dot matrix printers, but Magic Wand permits you to embed command codes for output directly to a printer, so a well-informed user can overcome the oversight. In addition to the usual line, margin, and character controls, you get your choice of nine degrees of boldface, broken or solid underline, support for true proportional spacing, and two-column printing. Headers or footers with page numbers can be run right, left, center, or alternating book style.

Magic Wand provides programmable, conditional and variable commands for creation of form documents to be merged with a data file (customer or mail list) for production of mass-produced "personalized" letters or other documents. Coupled with Magic Mailer, the excellent companion mail list manager, Magic Wand makes a powerful business resource.

The print formatting features are powerful, flexible, complex, and too big to cover here in detail. Just assume that one way or another you can make your letter-quality printer do anything it is capable of. However, most people will find the Print program not at all easy to master, nor easy to teach to new users. Trying to create a new or unusual format can be tricky and tedious. The saving feature is that favorite formats can be saved in a boilerplate file and called up for future use, and those will then be easy to use and easy to teach to new users. The video preview is useful but does not show at all what your hard copy will look like. Indents and centerings are accomplished with embedded commands, not by positioning the text on your screen, and bold face is shown with repeated characters, so **POW!** will appear as PPP OOO WWW !!! The print program would be greatly enhanced by hyphenation help, which it now lacks. In spite of its faults, Magic Wand can print sophisticated documents of high print quality on a letter-quality printer, being better in this regard than almost any other program. It is one of the few programs which will support two-column printing. Testimonials: Sorcim acquired the Print features of Magic Wand for their new wordprocessor, Superwriter. Using Magic Wand, I have published several books with galleys taken directly from my Diablo. It will make excellent business documents, such as catalogues, brochures, reports, proposals, flyers, prices lists, newsletters, and so on.

Taken on an individual basis, I have seen various edit and print features on other programs that are superior in numerous ways, but never in the same program, and never in a program which, taken overall, I would rather use regularly. Palantir could almost change my mind if I were not already so well settled in (Spellbinder would, too, if it would be made as well thought out and professionally finished as these two, but it isn't). How does Palantir compare to Magic Wand? Let's see.

Palantir

Preliminaries: The recently released Palantir was a fine program when it first hit the streets this year, and it has an even brighter future because of planned enhancements. This program is completely different from Magic Wand, yet it is of the same high quality or better throughout. It should be, because it was for the most part written by the same guy, Michael Griffin, who had such a big hand in writing Magic Wand. Two years ago, he and Bill Radding left SBA (the original MW owners) in the same cloud of smoke and teamed up to develop Palantir, which they wrote in C. They care about their program, keep improving it, and deliver considerate, concerned, and thoughtful support.

The documentation is unusually good, especially now that an index has been added. It is clear, complete, pleasant, and *brief!*, getting it all said in about 80 pages, not counting the index. If you too write any documentation, take a look at this manual. The high quality is partly due to writing skill, and partly due to Palantir being so logical and well thought out in the first place.

Palantir comes from the publisher with configuration files for 38 popular terminals and 18 printers, plus .ASM files for customization of your own drivers. Unlike Magic Wand, WordStar, et al, Palantir makes use of the built-in features of Epsons. A real innovation is the ability to switch between printers from within the program, allowing you to use a draft and a letter-quality printer. Five proportional print wheels are supported, and you get their .ASM files so you can configure the space table to your own taste and needs. Installation is easy and you can order a customization guide if you want to tinker. In sum, Palantir consists of various overlays adding up to 104K. Add 2K for each printer control file you need, plus 2K for proportional wheel control files, if needed. The help file, which I have never used, is 24K.

Palantir is an integrated program, meaning that text does not have to be saved before it is printed. Palantir initiates all wordprocessing functions from the main menu which comes up displaying your choices: Edit, Read, Save, Backup, File, Print, Type, Define, Help. The easiest way to initiate a choice is to merely press the first letter of your choice. You don't have to hit RETURN. Below the line of choices is a display of the status of the work in progress: the default drive, which drive has the work on it, the ID of your terminal, the name of your current file, the size of your file in characters (no word count), current location of cursor, % of disk used, and number of bytes left on the disk.

Three main menu choices need explaining. "File" brings up a submenu prompting choices for directory display: rename, erase, copy, display contents of a file, or change disks in a drive. "Define" will be explained below. "Type" is a wonderfully useful innovation which hooks your keyboard to your printer so you can use it as a typewriter.

Use of the screen is smooth and professional. You'll admire the way it looks and feels. For terminals that can support it, Palantir displays text in half intensity, using full intensity for highlighting. This gives a nice, soft screen which is easy on the eyes. However, those with Televideo 950s will find the 25th status line even more objectionable than usual, being brighter than the rest of your screen, but it can be turned off with the patch shown in the software tips section in this issue.

Editing: Pressing E gets you to the editing screen which is blank unless you have previously filled it with the Read function. The edit commands are illustrated in Figure 2 (see page 33), shown on function keys on the top line, and the control key equivalents for plain terminals are shown below. Unlike Magic Wand, the way the commands work is not obvious, but once they are explained, the logic is immediately apprehended and easily followed. Palantir includes labels which you can stick on your keys to remind you what and where the commands are.

The CANCEL key reverses commands and menu choices by backing out a step at a time down the same selection path you got in on. It will not, however, restore text which has just been erased, so it is not a true "Oh, $#)+!" key. The SET key and the CLEAR key do nothing alone, but are used in conjunction with other keys for a wide variety of functions, as you will see below. What SET-X initiates, CLEAR-X terminates.

The top line of your text screen contains status information: the location of the cursor, the insert/change mode, and an arrow to show in which direction certain commands will operate, and which is toggled with the DIRECTION key. This effects scrolling, search and replace, delete, and FIND. The first line of each text file is always the print ruler or format line which shows the margins, tabs, and justification mode.

Palantir is an on-screen formatter, showing you exactly how your hard copy will look (unless you use proportional spacing, in which case it is close but not exact), so the first thing you do in any new file is to define parameters by selecting Define in the main menu, if you want to change its default values. The define submenu prompts your choices for page length, top margin, bottom margin, print font, offset (left margin), single sheets or continuous forms, and beginning page number. Next, you enter the Edit mode and set the format line (print ruler) by pressing SET-FORMAT. This turns on the format menu, prompting you to set Margins, Tabs, Justification mode (Normal, Semi-justified, Justified, or Program-mode), or Other. If you choose Other, you get a submenu prompting you to set character spacing, lines per inch, line spacing, print wheel selection, boldface style (doublestrike or shadow), half or quarterline sub/superscripts, overstrike character, and the number of times to strike each character (which can be set at more than one for thick multiforms). The format line, all format choices, and the parameters set in the define menu are kept in a header record as a permanent part of the file. Only the format line is visible, so your screen is uncluttered. You can create format lines at any point in the text to vary any of these parameters.

Setting the formatting parameters takes longer to read about than to do, and once done, most or all of the work for formatting your printout is also done. Now you can begin work on your text.

Because Palantir formats on the screen, you may have to reformat your text after making inserts or deletes in the middle of an existing paragraph. This is a minor inconvenience, but it works smoother and faster in Palantir than in other programs. Either method you use for deleting anything larger than a word will automatically initiate reformatting of the paragraph. In any case, Palantir is very fast at reformatting.

The cursor keys work as you would expect them to. FIND <character> will search for that character in the direction indicated by the direction arrow. The scrolling keys move you a LINE, SCREEN, or PAGE in the direction of the arrow, or to the top or bottom of the DOCUMENT. Horizontal scrolling handles documents up to 250 characters wide. HOME

(continued on next page)

moves the cursor to the left of the current line, then to the top, then to the bottom of the screen. SET-HOME establishes a "home base" to which you can always return with FIND-HOME. SET-FIND initiates search and replace functions which are effective and flexible. You can search in any direction for character strings or whole words using "?" as a wildcard, making replacements, if desired, one at a time, n times, or all automatically. Word search ignores case, and the Replace string automatically maintains the same case arrangement as the Find string.

The Tab key runs the cursor to the marks set in the format line when in empty space, and to the beginning of words when in text. SET-TAB is used to establish a column of characters under the tab marks in the format line, so space between columns established with SET-TAB can be varied simply by moving the tab marks on the format line. The decimal tab works in much the same way, except that SET-DECIMALTAB organizes numerals (or characters) on the decimal mark. Other tab functions: SET ← centers text, and SET → runs the line flush to the right margin. SET-I (indent) establishes a temporary left margin at the cursor location.

The DELETE key erases the character under the cursor and moves text in from the right to fill the gap, the cursor remaining stationary. The BACKSPACE function key deletes the character to the left of the cursor, and moves the cursor and all text to the right of it one space to the left. Larger deletes are handled either with CLEAR-LINE which deletes all characters to the right of the cursor and reformats the paragraph, or with SET-DELETE which initiates rangefinding. After SET-DELETE, strike any character and the cursor runs to it, highlighting all text in between, which will be permanently deleted if you hit RETURN, and then reformatting the paragraph automatically. Thus, you can delete a word with SET-DELETE and a space, or a sentence with SET-DELETE and a period, or a paragraph with SET-DELETE and SET-RETURN. You can also just use the cursor or scroll keys to move the cursor to the end of whatever section you want deleted. This method works very nicely, but I have suggested that they give us at least one more function key which combines SET and DELETE so as to reduce the number of keystrokes.

Blocks can be moved, copied, named and saved, or deleted. SET-B marks the beginning of a block and initiates rangefinding which works exactly as described above for SET-DELETE functions. Unfortunately, there is no columnar block-move function.

The boilerplate function is smooth, effective, but definitely limited. You can make a lexicon with up to 36 entries coded A-Z and 0-9. Each entry can be up to 250 characters long. You can have more than one lexicon, but you can only use one per session, and switching between them is clumsy, as you have to rename two files before starting in order to make the switch. Displays of external files are smooth and efficient, and parts of them can be included but the process is clumsy, unlike the smoother Magic Wand.

File Handling: Unless you specify otherwise, Palantir automatically gives all wordprocessing files .WP as the extent, and backup files are given .WPB. You can specify any other extent you like, but Palantir will assume that the file is not for wordprocessing (either a program or data file) so will change its assumptions: no pagination, no wordwrap, it will not store the header record containing all the format settings, and tabs will be set for programming. This naming convention will annoy people who do a lot of wordprocessing and depend upon the extent to give order and meaning to their directories. Next time, we come to the rescue with a review of programs designed to overcome the bottleneck caused by CP/M-80's limitations in the directory department. Otherwise, Palantir's file handling is as excellent and reasonable as the rest of the program.

Error Trapping: Just as outstanding as Magic Wand, with one exception. When you change disks during a word session you *must* do so with the New Disk option in the main menu. Failure to do so, will cause an unrecoverable "BDOS Error: R/O" and you will lose all your work since the last save.

Printing: The first step in creating any new file, as described above, is to set Define and format line parameters, which leaves very little or nothing left to do in order to print the file. When you order Palantir to print, a menu comes up showing the print defaults, offering you an opportunity to make changes in the left margin, page on which to start, number of copies, single/continuous forms, and selection of printer driver. You can, of course, go back to Define to alter preset definitions, or go into the text to change format lines, at any time.

If you want to see where the pages will break, you can go into the text, put the cursor on the first line and press SET-P, which will insert a dotted line at each page break. You can force a page break at any location with SET-PAGE. Widows and orphans are automatically avoided unless you disable this function (widow: when the last line of a paragraph appears on next page; orphan: when the first line of a paragraph appears as the last line on the page). You can also keep any group of lines from being broken up and printed on two separate pages. This is great for charts, lists, and tables.

Headers, footers, and page numbers are very easy to set up, unlike with Magic Wand and other programs. They can be placed anywhere on the line, or occupy several lines, or be different on alternating pages. Just show Palantir what you want, where you want it, and "SET" it. Palantir supplies the full range of character control: boldface, shadow, underscore, double underscore, overstrike, strikeout, ribbon shift, normal and alternate print fonts, and extended print characters.

Files can be chained together for printing, or you can nest them by ordering the printing of one file from within another. You also have a broad range of conditional and variable commands for merging a document with a mail list or data file at print time. Palantir is compatible with almost any mail list program. Almost the only thing it lacks is two-column printing, and control over character spacing when printing proportionally.

Evaluation Summary

Both programs are excellent, so choosing between them is a matter of matching their relative strengths and weaknesses with the user's needs. This is a neck-and-neck horse race, but Palantir may win by a nose hair. In general, for most people and most purposes, Palantir is the better choice because Magic Wand's edge as an editor is strongly outweighed by the far easier, faster print functions in Palantir. Magic Wand is the choice if you need big boilerplate files, two-column printing, refined control over character spacing in proportional spaced printing, or if you expect to do lots of includes from external files. Magic Wand's slight superiority as an editor is primarily because all commands are a single keystroke, whereas Palantir requires four strokes for many deletes, and has numerous two-handed double-stroke commands. Magic Wand supports unlimited boilerplate files, and while Palantir's Lexicon is a better feature, it is limited to 36 entries. Including text from external files which are not boilerplated is much easier with Magic Wand. Palantir will scroll horizontally, Magic Wand will not. Palantir has disk-buffering to permit work on files larger than RAM, Magic Wand does not. Palantir will handle columns a bit better, but neither program has columnar block moves.

Take your choice, you won't go wrong with either one.

Also Ran

This is terribly brief, but in case I offended by omitting someone's favorite program, here's what happened. Perfect Writer, an otherwise competent program, was omitted here because it is too big, bulky, and complicated. It should be installed by a skilled technician who is also adept at word processing, and it is intimidating and overwhelming for most new users. Spellbinder has some truly brilliant features that I covet, but it is not well thought out and not professionally finished. The editor is clumsy, and the file handling is amateurish, just to name two items. Select is published by wonderful, sincere people, so I wish I could like it, but they spoiled it by inflexibly following their own logic. It is a limited program which is okay for short, infrequent use, but quite clumsy at editing larger documents.

In the future, if Final Word, Superwriter, Write, Metatype, or Benchmark turn out to be hot rivals of Palantir or Magic Wand, you will read about it here.

Figure 1. Keyboard Layout for Magic Wand Edit Commands

i) Simple keyboards:

Cursor up, down, left, right, backspace	cursor keys	Line forward	ctrl-X	Line delete	ctrl-N
		Line back	ctrl-E	Character insert	ctrl-v
Home	home key	Page forward	ctrl-C	Full insert	ctrl-O
Tab	tab key	Page back	ctrl-R	Search/replace	ctrl-G
Top of text	ctrl-T	Character delete	ctrl-D	Repeat search	ctrl-F
Bottom of text	ctrl-B	Word delete	ctrl-Y	Block marker	ctrl-U

ii) On Function Keys, Using the Televideo 950 as an example:

f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11
up *LINE* down	up *PAGE* down	end *TEXT* top	char. insert	char. delete	full insert	word delete	search / replace	rpt. srch	block mark	line delete
/——— white ———/			green	red	green	red	gold	gold	white	red

(Note: I added colored labels for fast, positive recognition. Notice that f1-f3 are used with and without the shift key.)

Figure 2 — Keyboard Layout for Palantir

a) on Televideo 950

Set	Clear	Insert	Direction	Format	Lexicon	find	Bkspc			Cancel	Line	Screen	Page	Doc.
↑S	↑C	↑V	↑Z	↑F	↑Q	↑G	↑B			ESC	↑E	↑R	↑T	↑D

b) on terminal without function keys (ADM-3A)

(continued on next page)

Software Notes — Tips and Techniques

Charles E. Sherman

Peach Thorns Removed From PeachText (Magic Wand)

When PeachTree bought Magic Wand, they acquired a really fine wordprocessor which they renamed "PeachText". Then they tinkered with it a bit, but not all their changes are appreciated. The new main menu is fine, especially since it is a separate file, "Menu.Com", which you can simply omit if you don't want to use it. The word count was omitted for no particular reason, and there's no fix for that. However, two of the really annoying little PeachThorns they installed can be fixed very easily simply by changing three bytes in Edit.Com and three in Print.Com.

The way PeachText is delivered from PeachTree, it defeats your efforts to create a file with no filetype (FTP), i.e., "CHAPTER1", or "MEMO", by automatically giving the FTP .DOC when no FTP is specified. The FTP .DOC has no function in the program whatever, so their reason for doing this is inexplicable. Stranger yet is the fact that if you have a file with no FTP which was created on some other program, when you name it in PeachText you have to put a period at the end of the filename, otherwise the program will not see it. This is completely contrary to CP/M naming convention, and causes nothing but confusion and lost time.

CP/M has notoriously limited directory naming and sorting capabilities, and with the larger capacity disks, directories become chaotic and difficult to use. I use the FTP extensively to make sense and order out of the directory, and I use files with no FTP to indicate the most active files or those most centrally relevant to the subject stored on that disk.

If you want to defeat the automatic .DOC FTP, and eliminate the need to put a period at the end of filenames with no FTP, then all you have to do is use DDT to change three bytes in Edit.Com and three in Print.Com. The list shows the bytes to be changed with their existing hex values. Change them all to 20 hex.

EDIT.COM		PRINT.COM	
39F5	44	3511	44
39F8	4F	3514	4F
39FB	43	3517	43

Thanks are due to PeachTree for furnishing this information. Unfortunately, they won't say how to fix another annoying PeachThorn, namely the numerous pauses in the program which flash the message, "Press RETURN to Continue", when there are no choices to make, nothing to enter, and no way to change the outcome of the next step, which usually returns you to the menu or CP/M. In other words, a pause and an extra keystroke with no useful purpose or meaning. If any fanatical disassemblers out there can figure out how to make this one go away, please write.

Turning Off The Televideo 950's 25th Line

One is not supposed to look a gift horse in the mouth, but when the gift is the 25th line on your Televideo 950, then you end up looking at it all the time whether you want to or not. It does get tiresome and distracting, and the problem is emphasized when you use a program that uses half-intensity for its main display, reserving full intensity for highlighting, intending to give the user a soft screen that is easy on the eyes. Unfortunately, the 25th line comes blasting through, and threatens not only your eyesight, but the phosphorus on your CRT. Whatever the reason, most users don't want that 25th line, but can't figure out how to make it go away. In the November, 1982 issue of *BYTE*, Jerry Pournelle writes:

> "Alas, it's nearly impossible to get rid of that line. It takes the darndest sequence of escape and control characters you ever saw because what you must do is fill a line with spaces and output all 80 of them. If there's a better way, neither Tony Pietsch nor I have been able to figure it out from the Televideo 950's rather poorly organized documents."

Lifelines flies to the rescue! Dear Byte, Jerry, and Tony: Here's how to make that bad old line go away. First, make sure your terminal has the bug-free firmware (2 PROMS) installed, version 2.0. If you type ESC-M <CR>, your terminal will display the firmware version number, plus the number of extra pages of memory installed. If it doesn't have revision level 2.0 or higher, write or call Televideo and they will send you free replacements. Now you have several different ways of turning off the 25th lines. You can simply type in this sequence from the keyboard: Shift-ESC f, ESC G0 <CR>, Shift-ESC g. Shazam! It's gone!

Of course, next time you power on, it will be back, and typing this sequence can become tiresome if you always want it off. Don't despair, there are two more easy ways to make it go away. Typing the above sequence is merely one way of getting the terminal to read a code sequence, which is only 7 bytes of hex, namely:

```
<1B><67><1B><66><1B><47><30>, or in ASCII
 ESC  g  ESC  f   ESC  G   0
```

Any way you can get the terminal to read this code will work. One way is to create a file which contains those seven characters named, say "TURNOFF", then at any time you are in CP/M you can order, "TYPE TURNOFF," and the job is done until the next poweroff. You can also turn TURNOFF into TURNOFF.COM and save keystrokes that way.

Well, for those like myself who want the line permanently off automatically, there is an even better solution. Put the seven bytes into the coldboot routine so every time you boot up, the 25th line gets the boot. This can be done properly and conventionally by editing your bios to include the seven characters, reassembling it, and reinstalling with SYSGEN. However, I have to confess that I did it " quick and dirty" by using DDT to patch it in as a substitute for seven characters in the signon message. Now my manufacturer's name no longer comes up when CP/M is cold booted, but rather my initials do, as a signal that this system has been tampered with. The missing 25th line is the only other incriminating evidence.

Opinion

Letter To The Editor

Some Comments on Money Maker's Forum

November 5, 1982

Dear Editor

The reviews of two Hayden titles that appeared on page 35 of the October issue in the "Money Maker's Forum" column by Charles E. Sherman were vindictive and unwarranted. In fact, the reviews amount to an outright attack on Hayden Book Company and the reputation of two of our authors. Permit me to present other views to set the record straight.

How to Profit from Your Personal Computer: Professional, Business, and Home Applications by Ted Lewis was published in 1978. In reviews that appeared at the time of publication, the book was praised as "A profitable acquisition for those seeking ways to put their computer to work"; another reviewer stated "Of all the books now available on personal computing, this is easily the best." Your reviewer's major delusion concerns the book's title, which in his mind is an "out-and-out misrepresentation and deception perpetrated by the publisher." His narrowly conceived notion of profit seems to be net income after costs. However, a reasonable reading of the title suggests that *Home to Profit* means *how to derive benefits or services from.* The cover illustration of George Washington accurately reflects the author's focus on financial application programs such as mortgage analysis, budgeting, and simple accounting systems.

Your reviewer is even more abusive in his pretentious diatribe on Joe Weisbecker's (not Weisbeck as appeared in the column) *Home Computers Can Make You Rich.* Your reviewer's excessive comments regarding the author are unconscionable. Mr. Weisbecker is a computer expert who holds 24 patents; he also designed the RCA COSMAC VIP single-board computer. Again, I quote from another published review: "The author's four basic ways to make money are... selling products relating to microcomputers...; selling services...; creating new products; and gambling (invest in small businesses; trade stock, etc....)... Weisbecker includes much common sense material... telling how others have made money in a variety of ways."

Hayden Book Company has earned the reputation of a quality book publisher — and the leading publisher of microcomputer books. Computer store owners and microcomputer users around the country will support that statement.

Although your reviewer is entitled to his personal opinions, a rebuttal is demanded when those opinions aim to damage the reputation of respected authors and companies under the guise of "informing computer consumers". It is difficult to believe that *Lifelines* finds his self-serving bombast to be the type of information that your subscribers are anxious to read.

Sincerely,

Michael Violano
Editorial Director
Hayden Book Company, Inc.
Rochelle Park, NJ

(Mr. Sherman may be responding in an upcoming issue.)

Editorial (continued from page 3)

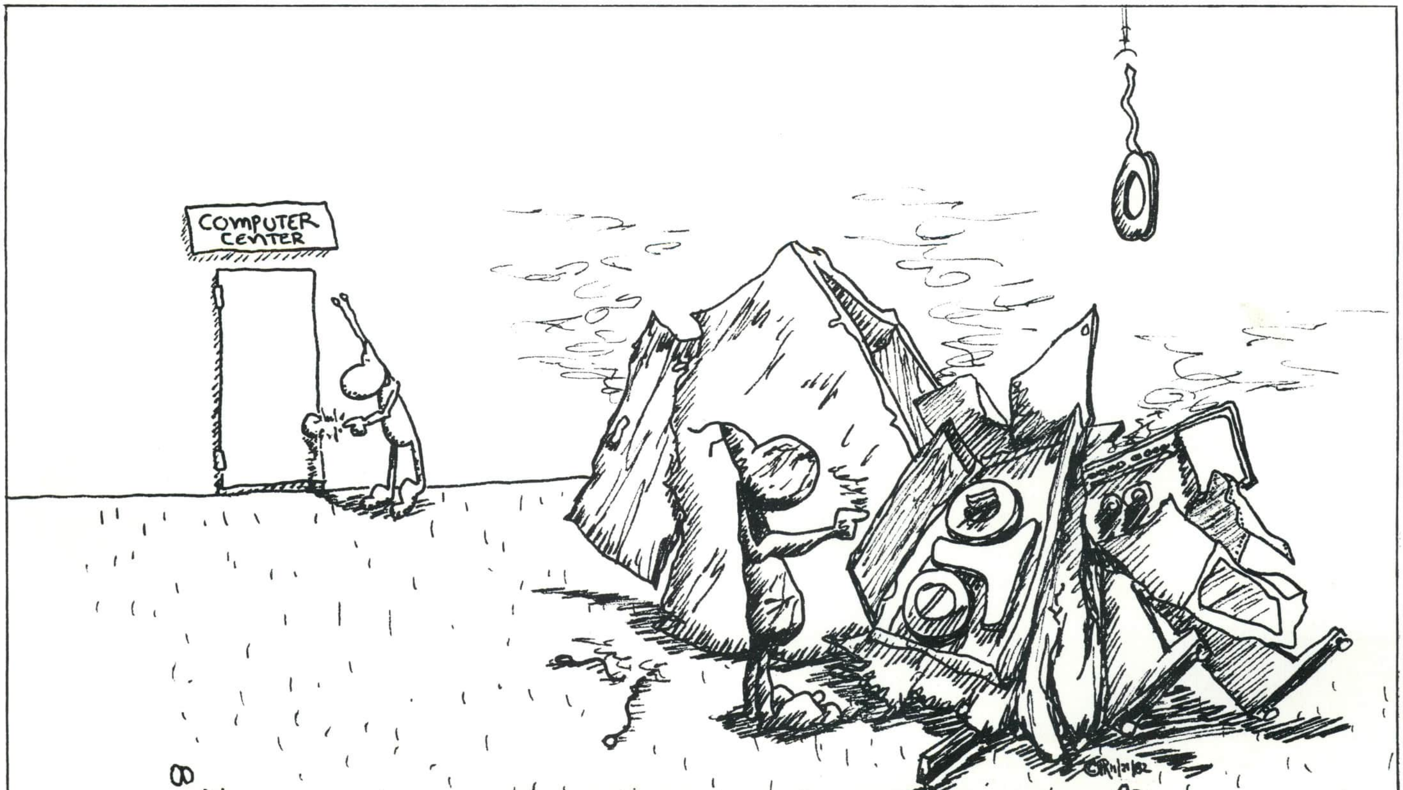
writing books on how to use languages and applications packages. One very interesting idea which has not yet been explored is the concept of detached keyboards which have no physical connection whatsoever. These units would be lined to the rest of the system via an RF link.

Nothing yet on flat CRT systems but keep watching – they are on their way.

It's rumored that the next COMDEX will be even bigger. If this trend continues it will be necessary for attendees to employ bicycles just to get through the show. It took me seven and a half hours to visit the booths at COMDEX this time and that was only possible by minimizing blabbing (i.e., unstructured talking) at each booth.

There's a new IBM PC publication coming called *PC World* with David Bunnell as publisher and editor-in-chief. This is his third PC-type publication. First *Personal Computing,* then *PC the Independent Guide to the IBM Personal Computer* and now *PC World.* One hundred thousand copies of the first edition should be available in January of 1983. Watch for it. David is one of those unreasonable men who doesn't compromise with excellence. It's bound to be a winner...

Lifelines (ISSN 0279-2575, USPS 597-830) is published monthly at a subscription price of $24 for twelve issues, when destined for the U.S., Canada, or Mexico, $50 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.



"YOU CAN REALLY GET SOME NEAT SPARKS OFF THIS CARPET!..."

LIFELINES™/The Software Magazine™
1651 Third Avenue, New York, New York 10028