SCP 86-DOS
Single-User Disk Operating System for the 8086

Introduction

86-DOS provides the tools needed to develop programs for the 8086, as well as a hardware-independent environment in which to run these programs. It is a very modular system. At its core is the disk file manager and I/O device handler, and everything else is considered a "user program". This allows the system to be easily tailored to any custom requirements.

The disk file manager allows programs to create and access disk files by name. Files may be read or written sequentially or randomly, any number of records at a time. File space on the disk is allocated dynamically, so that no "compaction" phase is ever required.

A program called COMMAND provides the interface between the user at the console and the file manager. COMMAND allows the user to display the disk directory, rename, destroy, or copy files, and execute other programs, such as the assembler, editor, or source code translator.

The assembler reads a source file of Intel-like 8086 mnemonics from disk, and produces a listing file and an Intel hex format object file. The program HEX2BIN may be used to convert the hex file to executable binary form.

The editor is line oriented, suitable for creating and maintaining program files. "Dynamic" line numbers, which are not actually present in the text, are used to identify lines to be listed, deleted, edited, etc. Global searching and global text replacement are also provided. After editing a file, the original file (before editing) is preserved as a back-up.

The source code translator can translate most Z80 source code into 8086 source code acceptable to the assembler after minor manual correction. This provides a relatively quick and easy way to transport programs between the processors.

A debugger is not yet provided with 86-DOS. The SCP 8086 Monitor provides good debugging aids in the interim.

SPECIAL NOTE: 86-DOS is not related to the popular CP/M operating system of Digital Research. Disk directory formatting and space allocation are completely different and incompatible. 86-DOS does, however, provide a utility called RDCPM which will transfer files from CP/M disks to 86-DOS disks. Further, operating system calls and calling conventions have been provided which make possible automatic translation of Z80 programs written for CP/M into 8086 programs that run under 86-DOS.

Operating System Calls

The purpose of the operating system core is to provide a high-level, hardware independent interface between a user program and its hardware environment. Most functions that the user may request can be grouped into two categories: simple device I/O and disk file I/O.

The simple I/O functions are:

Input console character
Output console character
Input from auxiliary
Output to auxiliary
Output to printer
Output character string to console
Input Buffered line from console
Check console for input character ready

The disk I/O functions include:

Reset disk system
Select default disk
Scan disk directory
Create file
Open file
Close file
Delete file
Rename file
Read/Write file record(s)
Set disk transfer address

Interrupt Table Usage

The first 1K of memory, absolute address 00000 to 003FF hex, is reserved by the 8086 for the interrupt table. Within this table, locations 00080 to 000FF, which correspond to interrupt types 20 to 3F hex, are reserved for 86-DOS. Specific interrupt types have been defined as follows (types in hex):

20 - Program terminate. The terminate and Ctrl-C Exit addresses are restored to the values they had on entry to the terminating program. All file buffers are flushed, but files which have been changed in length but not closed will not be recorded properly in the disk directory. Control transfers to the Terminate address.

21 - Function request. See "Requesting a Function", below.

22 - Terminate address. On entry to a program, this is the address to which control will transfer when the program terminates. This address is copied into low memory in the program segment when it is created by Function 38. A program may change this address, but this does not affect what happens when it terminates, since the Terminate address is restored from the copy in the program segment. If the program executes a second program, it must set the Terminate address to the location that the second program will transfer to on termination. If this is all clear to you, you will have no trouble understanding the rest of this manual.

23 - Ctrl-C Exit address. If the user at the console types Ctrl-C during console input or output, an interrupt type 23 hex is executed. If the Ctrl-C routine preserves all registers, it may end with a return-from-interrupt instruction (IRET) to continue program execution. If the Ctrl-C handler does nothing but an "IRET", Ctrl-C will appear to have no effect.

If the program executes a second program which itself changes the Ctrl-C Exit address, then on termination of the second program and return to the first, the Ctrl-C address is restored to value it had before the second program changed it.

25 - Absolute disk read. Control transfers directly to the I/O system disk read routine. On return, the original flags are still on the stack (put there by the INT instruction), which is necessary because return information is passed back in the flags. Be sure to pop the stack to prevent uncontrolled growth.

26 - Absolute disk write. See above.

Requesting a Function

The user program requests a function by putting a function number in the AH register, possibly setting another register according to the function specifications, and performing an interrupt type 21 hex. When 86-DOS takes control it switches to an internal stack and saves all the user's registers except AL. Thus the user's stack need only enough space to perform the interrupt (6 bytes), and all registers, including the flags but excepting AL, will be unchanged on return unless noted otherwise in the function specification.

The Standard Functions (as opposed to the Extended Functions), whose function numbers are 36 or less, are also available through a different call mechanism. The function number is placed in the CL register, any other registers are set in their usual way according to he function specifications, and a normal (intra-segment) "call" is made to location 5 in the current code segment. Register AX is always destroyed by this calling method, but otherwise it is the same as the normal (interrupt) method. This form is provided to simplify translation of 8080/Z80 programs into 8086 code, and is not recommended for new programs.

Simple Device I/O Functions

1 - Console input. Waits for a character to be typed on the console, then echos the character (as in Function 2) and returns it in AL. The character is checked for a control function as described in Function 2 below.

2 - Console output. The character in register DL is output to the console. Control characters other than carriage return, linefeed, backspace, and tab are displayed as an up-arrow followed by the associated letter. Tabs are expanded in columns of 8. Rubout (7F hex) is output but is not counted in tab counting. After output, the console is checked for a control function:

Ctrl-S suspends everything until any key is typed.

Ctrl-P sends all console output to the printer also.

Ctrl-N stops sending output to the printer.

Ctrl-C causes an interrupt to the Ctrl-C address.

3 - Auxiliary input. Waits for a character from the auxiliary input device, then returns that character in AL.

4 - Auxiliary output. The character in DL is output to the auxiliary output device.

5 - Printer output. The character in DL is output to the printer.

6 - Direct Console I/O. If DL is FF hex, then AL returns with a console input character if one is ready, otherwise 00. If DL is not FF hex, then DL is assumed to have a valid character which is output to the console.

9 - Print string. On entry, DS:DX must point to a character string in memory terminated by a "$" (24 hex). Each character in the string will be output to the console in same form as Function 2, including subsequent status check.

10 - Buffered console input. On entry, DS:DX point to an input buffer. The first byte must not be zero and specifies the number of characters the buffer can hold. Characters are read from the console and placed in the buffer beginning with the third byte. Reading the console and filling the buffer continues until carriage return is typed. If the buffer fills to one less than maximum, then additional console input is ignored until a carriage return is received. The second byte of the buffer is set to the number of characters received excluding the carriage return (OD hex), which is always the last one.

A number of control functions are recognized while reading the console:

Tab, Ctrl-S, Ctrl-P, Ctrl-N, Ctrl-C have the same effects as listed under Function 2.

Rubout, delete, backspace, Ctrl-H (7F hex or 08 hex): Backspace. Removes the last character from the input buffer and erases it from the console.

Linefeed, Ctrl-J (10 hex): Physical end-of-line. Outputs a carriage return and linefeed but does not effect the input buffer.

Ctrl-X (18 hex): Cancel line. Outputs a back slash, carriage return, and linefeed and resets the input buffer to empty.

SPECIAL EDITTING COMMANDS. A number of special editing commands are available to the user entering a line at the console. All of these involve a "template", which is a valid input line available to the user for modification. There are two ways to obtain a template.

If the input buffer already contained a valid input line on entry to Function 10, then this is a template. A valid input line is one in which the character count at the second byte of the buffer is less than the buffer length, and a carriage return (OD hex) immediately follows the text in the buffer. Note that a buffer that has previously been used for input and has not been modified will meet these requirements.

The user at the console may also create a template. One of the editing commands is to convert that part of the line entered so far into the template, and restart the line entry. This allows an error near the start of a line to be corrected without retyping the rest of the line.

Each editing command is selected by typing ESCAPE and a letter. Since many terminals provide keys which produce such an "escape code" with a single keystroke, the letter used after the ESCAPE may be set for each command during 86-DOS customization. The standard escape sequences correspond to the special function keys of a VT-52 or similar terminal, as noted in each case by parentheses.

ESC S (F1) - Copy one character from the template to the new line.

ESC T (F2) - Must be followed by any character. Copies all characters from the template to the new line, up to but not including the next occurrence in the template of the specified character. If the specified character does not occur, nothing is copied to the new line.

ESC U (F3) - Copy all remaining characters in the template to the new line.

ESC V (F4) - Skip over one character in the template.

ESC W (F5) - Must be followed by any character. Skips over all characters in the template, up to but not including the next occurrence in the template of the specified character. If the specified character does not occur, no characters are skipped.

ESC P (BLUE) - Enter insert mode. As additional characters are typed, the current position in the template will not advance.

ESC Q (RED) - Exit insert mode. The position in the template is advanced for each character typed. When editing begins, this mode is selected by default.

ESC R (GRAY) - Make the new line the template. Prints an "@", a carriage return, and a line feed. Buffer is set to empty and insert mode is turned off.

11 - Check console status. If a character is waiting at the console, AL will be FF hex on return. Otherwise, AL will be 00.

Disk I/O Functions

Disk files are identified by a disk drive code, a file name of up to 8 characters, and an extension of up to 3 characters. The drive code may explicitly specify a drive, or the default drive may be used. Case is irrelevent in the file name or extension, since only upper case is used internally. If the file name or extension includes a question mark ("?") in any position, then that position will match any character. Thus a single file name with embedded question marks may match more than one directory entry.

Generally, functions operating on disk files will use a File Control Block, or FCB. The FCB is a 33- or 35-byte segment of memory with information about a file, formatted as follows:

BYTE 0 - Drive Code. Zero specifies the default drive, 1=drive A, 2=drive B, etc. Note that other functions which use a drive number use 0=drive A, 1=drive B, etc.

BYTES 1-8 - File Name. If the file name is less than 8 characters, the name must be left justified with trailing blanks.

BYTES 9-11 - Extension. If less than 3 characters, must be left justified with trailing blanks. May also be all blanks.

BYTES 12-13 - Current Block. This word (low byte first) specifies the current 16K block, relative the start of the file, in which sequential disk reads and writes occur. If zero, then the first 16K of the file is being accessed; if one, then the second 16K; etc. Combined with the current record field, byte 32, a particular 128-byte record is identified.

BYTES 14-31 - Reserved for system use once the file is opened and until it is closed.

BYTE 32 - Current Record. Identifies the record within the current 16K block that will be accessed with a sequential read or write function.

BYTES 33-34 - Random Record. This word (low byte first) need be present only when the file is accessed with a random read or write function. This 16-bit number is the position in the file of a 128-byte record.

Notice that there are two ways to address a record within a file. The Current Block and Current Record fields together address a record when the file is accessed with the sequential read and write functions. The Random Record field addresses a record when the file is accessed with the random read and write functions. The appropriate fields may be set before either a sequential or random transfer to select the next record to be accessed.

An unopened FCB is one in which only the first 12 bytes have been filled in, i.e., name and drive code. An opened FCB is one that has been through a successful open or create operation (Functions 15 or 22) and has its Random Record or Current Block/Current Record fields set as necessary.

13 - Disk reset. Selects drive A as the default drive, sets the disk transfer address to DS:80 hex, and flushes all file buffers. Files which have been changed in size will not be properly recorded in the disk directory until they are closed. This function need not be called before a disk change if all files which have been written to are closed.

14 - Select disk. The drive specified in DL (0=A, 1=B, etc.) is selected as the default disk.

15 - Open file. On entry, DS:DX point to an unopened FCB. The disk directory is searched for the named file and AL returns FF hex if it is not found. If it is found, AL will return a 00 and the FCB is filled as follows:

If the Drive Code was zero (default disk), it is changed to actual disk used (A=1, B=2, etc.). This allows changing the default disk without interfering with subsequent operations on the file.

The Current Block field is set to zero.

All remaining fields, up to but not including the Current Record field, are filled with system information. It is the calling program's responsibility to set the Current Record or Random Record fields as necessary.

16 - Close file. This function must be called after file writes to insure all directory information is updated. On entry, DS:DX point to an opened FCB. The disk directory is searched and if the file is found, its position is compared with that kept in the FCB. If the file is not found in its correct position in the directory, it is assumed the disk has been changed and AL returns FF hex. Otherwise, the directory update is completed and AL returns 00.

17 - Search for first entry. On entry, DS:DX point to an unopened FCB. The disk directory is searched for the first matching name and if none are found, AL returns FF hex. Otherwise, the first 16 bytes at the current disk transfer address are filled with the dirctory entry and AL returns 00. The first 11 bytes are the 8-character file name and 3-character extension.

18 - Search for next entry. After Function 17 has been called and found a match, Function 18 may be called to find the next match in the directory. Additional matches will be found because of duplicate names or because of "?"s appearing in the file name. Return information is the same as Function 17. Since the file name was copied into an internal buffer by Function 17, no parameters are required, but also no intervening disk operations may occur between a call to Function 18 and the previous call to Function 17 or 18.

19 - Delete file. On entry, DS:DX point to an unopened FCB. All matching directory entries are deleted. If no directory entries match, AL returns FF, otherwise AL returns 00.

20 - Sequential read. On entry, DS:DX point to an opened FCB. The 128-byte record addressed by the Current Block and Current Record fields is loaded at the disk transfer address, then the record address in incremented. If end-of-file in encountered, AL returns 01, otherwise AL returns 00.

21 - Sequential write. On entry, DS:DX point to an opened FCB. The 128-byte record addressed by the Current Block and Current Record fields is written from the disk transfer address, then the record address in incremented. If the disk is full, AL returns 01, otherwise AL returns 00.

22 - Create file. On entry, DS:DX point to an unopened FCB. The disk directory is searched for an empty entry, and AL returns FF hex if none is found. Otherwise, the entry is initialized to a zero-length file, the file is opened (see Function 15), and AL returns 00.

23 - Rename file. On entry, DS:DX point to a modified FCB which has a drive code and file name in the usual position, and a second file name starting 6 bytes after the first (DS:DX+17) in what is normally reserved area. Every matching occurence of the first file name is changed to the second name. If question marks (3F hex) appear in the second file name, then the corresponding positions in the original name will be unchanged. AL returns FF hex if no match was found, otherwise 00.

25 - Current disk. AL returns with the code of the current default drive (0=A, 1=B, etc.).

26 - Set disk transfer address. The disk transfer address is set to DS:DX.

27 - Allocation table address. On return, DS:BX point to the allocation table for the current drive, DX has the number of allocation units, and AL has the number of records per allocation unit. This function is intended only for system utilities written by SCP.

31 - Disk parameter address. On return, DS:BX point to an internal table of parameters for the current default disk. This function is intended only for system utilities written by SCP.

33 - Random read. On entry, DS:DX point to an opened FCB. The Current Block and Current Record are set to agree with the Random Record field, then the 128-byte record addressed by these fields is loaded at the disk transfer address. If end-of-file is encountered, AL returns 01, otherwise AL returns 00.

34 - Random write. On entry, DS:DX point to an opened FCB. The Current Block and Current Record are set to agree with the Random Record field, then the 128-byte record addressed by these fields is written from the disk transfer address. If the disk is full, AL returns 01, otherwise AL returns 00.

35 - File size. On entry, DS:DX point to an unopened FCB. The disk directory is searched for the first matching entry and if none is found, AL returns FF hex. Otherwise the Random Record field is set with the size of the file (in 128-byte records) and AL returns 00.

36 - Set Random Record field. On entry, DS:DX point to an opened FCB. This function sets the Random Record field to the same file address as the Current Block and Current Record fields.

39 - Random block read. On entry, DS:DX point to an opened FCB, and CX contains a record count which must not be zero. The specified number of records are read from the file address specified by the Random Record field into the disk transfer address. If end-of-file is reached before all records have been read, then AL returns 01. If wrap-around above address FFFF hex in the disk transfer segment would occur, as many records as possible are read and AL returns 02. If all records are read successfully, AL returns 00. In any case, CX returns with the actual number of records read, and the Random Record and Current Block/Current Record fields are set to address the next record (the first record NOT read).

40 - Random block write. Essentially the same as Function 39 above, except for writing. If there is insufficient space on the disk, AL returns 01 and no records are written. If CX is zero on entry, then no records are written, but the file is truncated at that point, i.e., all records beginning with the one addressed by the Random Record field are released to free space.

Miscellaneous Functions

0 - Program terminate. The Terminate and Ctrl-C Exit addresses are restored to the values they had on entry to the terminating program. All file buffers are flushed, but files which have been changed in length but not closed will NOT be recorded properly in the disk directory. Control transfers to the Terminate address.

37 - Set vector. The interrupt type specified in AL is set to vector to the address DS:DX.

38 - Create new program segment. On entry, DX has the segment number at which to set up a new program segment. The entire 100 hex area at location zero in the current program segment is copied into location zero of the new program segment. The memory size information at location 6 is updated, and the current Terminate and Ctrl-C Exit addresses are saved in the new program segment starting at 0A hex.

Running a User Program

The operating system core provides no direct means to run user programs. Instead, to run a given program represented by a disk file, the file must be opened and read into memory using the normal system functions. These functions are requested by the user program that is currently running.

The first user program to run is the initialization routine that follows a system boot, which normally loads and executes the file COMMAND.COM. This is a user program that accepts commands from the console and translates them into system function calls. COMMAND includes the capability to load and execute other program files; when these other programs terminate, COMMAND regains control. Thus COMMAND is responsible for the initial conditions that are present when a program is executed.

A standard set of initial conditions is provided by COMMAND on entry to another program. It is possible for programs other than COMMAND to load and execute program files, and they must also provide the same initial conditions so that a consistent interface may be assumed by the newly executing program. These initial conditions are as follows:

All four segment registers have the same value, and the corresponding absolute memory address is the base of a "program segment". The program is loaded and begins execution at location 100 hex in the program segment. Other assignments in the program segment are:

00 - 01: Termination point. Contains an interrupt type 20 hex, which returns control to the originating program. Thus a JMP 0 or INT 20H are the normal ways to terminate a program.

02 - 03: Memory size. Contains the first segment number after the end of memory.

05 - 05: Standard Function request entry point. See "Requesting a Function".

06 - 07: Segment size. This is the number of bytes available in the program segment.

08 - 09: Reserved.

0A - 0D: Terminate address. The address, in displacement:segment format, to which control will transfer on termination.

0E - 11: Reserved.

12 - 3F: Default stack. The stack pointer is initially 3E hex, with a word of zeros on the top. Thus executing a "return" instruction will cause a transfer to location 0 and the program will terminate normally. This stack may be used as-is, or a new one may be set up.

5C - 67,
6C - 77: Formatted parameters. Each of these areas may contain a parameter, usually a file name. The first byte of each area is zero unless a disk drive is being specified, in which case 1=drive A, 2=drive B, etc. The rest is blanks if no parameter is present. No lower-case letters are allowed in these fields--they must be converted to upper case. If the parameter is a file name, then the next 8 bytes have the name, followed by the 3-character extension. Thus each parameter is properly formatted as an unopened FCB, except that the reserved area of the first overlaps onto the second. If both parameters are used as file names, the second one must be moved to a different area or it will be destroyed when the first is opened.

80 - FF: Unformatted parameters. Any information to be passed may be placed in this area. The disk transfer address is initially set to 80 hex.

COMMAND prepares the parameter areas from the console input line that specified the program to be executed. For example, if COMMAND sees an line of the form

<progname> <file1> <file2>

this is a request to execute the file <progname>. <file1> and <file2> each may or may not include a disk specifier or a file name extension, but in any case they appear in the formatted parameters at 5C hex and 6C hex. In addition, the entire input line after the last letter of <progname> appears in the unformatted parameter area beginning at 81 hex, with the number of characters placed at 80 hex.

Suppose the input line is

```
COPY T.BAK B:TEST.ASM
```

The formatted parameter at 5C hex will contain

```
00 "T       BAK"
```

at 6C hex will be

```
02 "TEST    ASM"
```

and at 80 hex will be

```
17 " T.BAK B:TEST.ASM"
```

where the 17 is decimal.

Using Operating System Functions

Disk File Reading and Writing

It is strongly recommended that all disk I/O use the block read and block write functions, Functions 39 and 40, rather than Functions 20, 21, 33, or 34. Since the block read and write functions update the Random Record field of the FCB, they may be used for sequential access as well as random, or any intermixing. Programs which would ordinarily sequentially read or write one record at a time might experience considerable improvement in performance if several records were buffered instead of just one. The block I/O functions allow this buffer size to be variable, depending, for example, on available memory size.

The Line Editor: Function 10

The most straighforward use of the editing features provided by Function 10, buffered console input, is allowing the user to correct mistakes in the line currently being entered. However, their are two other important uses, both of which take advantage of the fact that a template may already be present in the input buffer before the system call is made.

The simpler of the two is used by COMMAND, which processes user commands. By simply re-using the same buffer each time an input line is requested, then the previous line entered becomes the template for the new line. This allows the user to easily repeat a command, or to correct an error in the previous command. Or when used with a BASIC interpreter, for example, the user could correct the last program line entered (since the line number insures the old line will be replaced), or the line number could be changed so that several similar lines could be entered easily.

If the program wishes to actively use the editing features, it may load any arbitrary text into the buffer before requesting Function 10. Note that the second byte of the buffer must be set with the character count and an ASCII carriage return must immediately follow the text in the buffer. EDLIN, the text editor provided with 86-DOS, uses this method to provide editing within a line. A BASIC interpreter with an EDIT command could load the specified line into the buffer and let Function 10 do the rest. Any program in which there is a "typical response" at a given moment could make the template this response to allow the user to select it easily.

It is important for any program that wishes to provide line editing to use the features of Function 10 to do so. This provides the user with a set of editing operations that are consistent from program to program, and that have been tailored in one step to match the user's terminal (during 86-DOS customizing).

CUSTOMIZING THE I/O SYSTEM

In order to provide the user with maximum flexibility, the disk and simple device I/O handlers of 86-DOS are a separate subsystem which may be configured for virtually any real hardware. This I/O system is located starting at absolute address 400 hex, and may be any length. The DOS itself is completely relocatable and normally starts immediately after the I/O system.

Beginning at the very start of the I/O system (absolute address 400 hex) is a series of 3-byte jumps (long intra-segment jumps) to various routines within the I/O system. These jumps look like this:

```
0000    JMP     INIT    ; System initialization
0003    JMP     STATUS  ; Console status check
0006    JMP     CONIN   ; Console input
0009    JMP     CONOUT  ; Console output
000C    JMP     PRINT   ; Printer output
000F    JMP     AUXIN   ; Auxiliary input
0012    JMP     AUXOUT  ; Auxiliary output
0015    JMP     READ    ; Disk read
0018    JMP     WRITE   ; Disk write
001B    JMP     FLUSH   ; Empty disk buffers
```

The first jump, to INIT, is the entry point from the system boot. All the rest are entry points for subroutines called by the DOS. Inter-segment calls are used so that the code segment is always 40 hex (corresponding to absolute address 400 hex) with a displacement of 3, 6, 9, etc. Thus each routine must make an inter-segment return when done (RET L with our assembler).

The function of each routine is as follows:

INIT - System initialization

Entry conditions are established by the system bootstrap loader and should be considered unknown. The following jobs must be performed:

A. All devices are initialized as necessary.

B. A local stack is set up and DS:SI are set to point to an initialization table. Then an inter-segment call is made to the first byte of the DOS, using a displacement of zero. For example:

```
MOV     AX,CS   ; Get current segment
MOV     DS,AX
MOV     SS,AX
MOV     SP,STACK
MOV     SI,INITTAB
CALL    0,DOSSEG
```

The initialization table provides the DOS with information about the disk system. The first byte is the number of drives (16 or fewer), followed by two 2-byte entries for each drive. The first of the two entries for each drive is the address (in the same data segment) of a disk drive parameter table (DPT). Similar drives may point to the same DPT.

Below is a brief description of each entry of the DPT. A more complete description with instructions for making DPTs for different types of disks will be available soon.

1. Number of 128-byte records per physical sector. 1 byte.

2. Number of 128-byte records per allocation unit. 1 byte.

3. Number of reserved 128-byte records at beginning of disk. 2 bytes.

4. Size of allocation table, in 128-byte records. Each allocation unit (item 7) requires 1.5 bytes in the allocation table. 1 byte.

5. Number of allocation tables kept on the drive. 1 byte.

6. Number of 128-byte records devoted to the directory. There are 8 directory entries per record. 1 byte.

7. Number of allocation units on the drive. 2 bytes.

The second of the two entries for each drive is the displacement of the allocation table for that drive. Normally, the first drive will be given a displacement of zero, and each subsequent drive will be assigned a space immediately after the previous drive's table ends. The size of the table for any drive is 128 bytes times the number of records specified in item 4 above. Note that no space need be provided by the I/O system for the allocation tables; this space is assigned by the DOS during initialization.

Below is a sample of a complete initialization table for four single-density IBM format disk drives:

```
INITTAB:
        DB      4       ; Number of drives
        DW      DRIVE0
        DW      AT0
        DW      DRIVE1
        DW      AT1
        DW      DRIVE2
        DW      AT2
        DW      DRIVE3
        DW      AT3

DRIVE0:
DRIVE1:
DRIVE2:
DRIVE3:
; All drives are defined the same
        DB      1       ; Records/sector
        DB      4       ; Records/allocation unit
        DW      52      ; Reserved records (two tracks)
        DB      6       ; Allocation table size, records
        DB      2       ; Number of allocation tables (1 backup)
        DB      8       ; Number of directory records (64 entries)
        DW      482     ; Number of allocation units (512 bytes ea.)

        ORG     0       ; Allocation tables are in their own segment

AT0:    DS      300H    ; Six 128-byte records
AT1:    DS      300H
AT2:    DS      300H
AT3:    DS      300H
```

C. When the DOS returns to the INIT routine in the I/O system, DS has the segment of the start of free memory, where a program segment has been set up. The remaining task of INIT is to load and execute a program at 100 hex in this segment, normally COMMAND.COM. The steps are:

1. Set the disk transfer address to DS:100H.

2. Open COMMAND.COM. If not on disk, report error.

3. Load COMMAND using the block read function (Function 39). If end-of-file was not reached, or if no records were read, report an error.

4. Set up the standard initial conditions and jump to 100 hex in the new program segment.

An example of code which performs this task is given:

```
        MOV     DX,100H
        MOV     AH,26
        INT     21H             ;Set transfer address to DS:100H
        MOV     BX,DS           ;Save segment for later
; DS must be set to CS so we can point to the FCB
        MOV     AX,CS
        MOV     DS,AX
        MOV     DX,FCB          ;File Control Block for COMMAND.COM
        MOV     AH,15
        INT     21H             ;Open COMMAND.COM
        OR      AL,AL
        JNZ     COMERR          ;Error if file not found
        MOV     [FCB+33],0      ;Set Random Record field
        MOV     CX,200H         ;Load maximum records
        MOV     AH,39
        INT     21H             ;Block read
        JCXZ    COMERR          ;Error if no records read
        CMP     AL,1
        JNZ     COMERR          ;Error if not end-of-file
        MOV     DS,BX           ;All segment reg.s must be the same
        MOV     ES,BX
        MOV     SS,BX
        MOV     SP,40H          ;Stack must be 40 hex
        XOR     AX,AX
        PUSH    AX              ;Put zero of top of stack
        MOV     DX,80H
        MOV     AH,26
        INT     21H             ;Set transfer address to default
        PUSH    BX
        MOV     AX,100H
        PUSH    AX
        RET     L               ;Jump to COMMAND

COMERR:
        MOV     DX,BADCOM
        MOV     AH,9
        INT     21H             ;Print error message
STALL:  JMP     STALL           ;Don't know what to do

BADCOM: DB      13,10,"Bad or missing Command Interpreter",13,10,"$"

FCB:    DB      1,"COMMAND COM"
```

STATUS - Console input status

If a character is ready at the console, this routine returns a non-zero value in AL and the zero flag is clear. If no character is ready, AL returns zero and the zero flag is set. No registers other than AL may be changed.

CONIN - Console input

Wait for a character from the console, then return with the character in AL. No other registers may be changed.

CONOUT - Console output

Output the character in AL to the console. No registers may be affected.

PRINT - Printer output

Output the character in AL to the printer. No registers may be affected.

AUXIN - Auxiliary input

Wait for a byte from the auxiliary input device, then return with the byte in AL. No other registers may be affected.

AUXOUT - Auxiliary output

Output the byte in AL to the auxiliary output device. No registers may be affected.

READ - Disk read
WRITE - Disk write

On entry,

AL = Drive number (starting with zero)
AH = Directory flag (WRITE only)
CX = Number of 128-byte records to transfer
DX = Logical record number
DS:BX = Transfer address.

The number of records specified are transfered between the given drive and the transfer address. "Logical record numbers" are obtained by numbering each record sequentially starting from zero, and continuing across track boundaries. Thus for standard floppy disks, for example, logical record 0 is track 0 sector 1, and logical record 53 is track 2 sector 2. This conversion from logical record number to track and sector is done simply by dividing by the number of records per track. The quotient is the track number, and the remainder is the record on that track. (If the first sector on a track is 1 instead of 0, as with starndard floppy disks, add one to the remainder.)

"Sector mapping" is not used by this scheme, and is not recommended unless contiguous sectors cannot be read at full speed. If sector mapping is desired, however, it may be done after the logical record number is broken down into track and sector. The 8086 instruction XLAT is quite useful for this mapping.

All registers except the segment registers may be destroyed by these routines. If the transfer was successfully completed, the routines should return with the carry flag clear. If not, the carry flag should be set, and CX should have the number of records remaining to be transfered (including the record in error).

On disk writes only, register AH is zero for normal writes and non-zero for directory writes. Thus if disk I/O is being buffered in memory, as would be the case if physical sector size is greater than 128 bytes, then this memory buffer must be flushed to disk when AH is non-zero to insure the directory is updated.

FLUSH - Empty disk buffers

This routine is called when a file is closed or when the disk system is reset. It may be used to write to disk any disk buffers that have been kept in memory. On entry, AL has the drive number whose buffers should be flushed, or if AL = -1, then flush all buffers. All registers may be destroyed except the segment registers. If memory buffering is not used, this routine may simply return (inter-segment).