Introduction

What is DOS?

The IBM Personal Computer Disk Operating System (DOS) is a collection of programs designed to make it easy for you to:

- create and manage files
- run programs
- simplify using the devices (for example, printer and disk drives) attached to your computer.

What is a file?

Nearly every business office has one or more filing cabinets containing folders of information. Usually, all the information in a given folder is related—for example, one folder might contain the names and addresses of all employees. Such a collection of related information is called a file, and can be given a name for easy reference.

All the information contained on your DOS diskette (text, programs, etc.) resides in files, each with a unique name, and any data you enter and save is also kept in files.

How many files can I have?

Each diskette can contain up to 64 files of varying sizes. If your files are large, the diskette may fill up with fewer than 64 files.

How are files kept track of?

The names of your files are kept on the diskette in a system area known as the **directory**, along with other pertinent information such as the size of the file, its location on the diskette, and the date it was created.

The directory occupies four sectors at a specific location on each diskette. (See the "Using DOS" chapter in the *Guide to Operations*.)

Next to the directory is a system area known as a File Allocation Table. Its function is to keep track of which sectors belong to which file, and to keep track of all available space on the diskette, so you can create new files.

Each diskette has one directory and two copies of the File Allocation Table. (If there is a problem reading the first copy of the File Allocation Table, the system uses the second one.)

Why are you telling me all this?

The information just described is important to you because these system areas are required on all diskettes that DOS is expected to recognize (not just the DOS diskette but on yours as well). There is only one way to get this information on a new diskette—a program called FORMAT which is supplied with DOS.

The first thing you must do with a new diskette is to format it with the FORMAT command (See Chapter 2.) which runs the FORMAT program. It writes on every sector of your diskette, initializes the directory and File Allocation Table, and puts a very important program, called the "boot record" at the very beginning of your diskette.

FORMAT also creates a copy of DOS on the new diskette if you ask it to. That way, you can create a diskette containing DOS and have plenty of room for your own data on the same diskette. Keep in mind that only the DOS files are copied—none of the other files on your DOS diskette are copied by FORMAT.

What can I call my file?

Diskette filenames are 1-8 characters long, and may be immediately followed by a filename extension of 1-3 characters. With few exceptions, you can give your files any names you want. (See Chapter 2.)

A few words about backups

It is strongly recommended that you make backup copies of all your diskettes, so that if a diskette somehow becomes damaged, or if files are accidentally erased, you will still have all your information. There are two ways to create a backup diskette (Remember to format it first.):

- Use the DISKCOPY command (See Chapter 2.) to create an exact image of an entire diskette on another diskette. You can use this command to copy the DOS diskette or your own diskette. The advantages of DISKCOPY are that it is the fastest way of copying a diskette, and that it copies everything, including DOS if it exists, in one operation.

  However, if either diskette has defective tracks, or if the diskette you want to copy has had a large amount of file creation/erasure activity, the method below is recommended because it compensates for the random placement of data caused by the creation/erasure activity.

- Use the COPY command (See Chapter 2.) to copy all files to the new diskette. This is slower, but it produces the same end result, with one difference– all your files will be written one right after the other (sequentially), which will "straighten out" the files, resulting in better performance if the diskette you are copying from had been subject to a lot of creation/erasure activity.

  If you use this method and want the new diskette to contain a copy of DOS, remember to tell the FORMAT command when you format the diskette because the system must reside on the diskette before any other files are placed on it.

  Note: In either case, the dates stored in the directory for each file are unaffected by copying.

What are the parts of DOS?

These three programs are the "heart" of your DOS.

1. The "boot record." This special program resides at the beginning of your diskette. It is automatically loaded into memory when you start up your system, and is responsible for loading the rest of DOS. It is placed on all diskettes by the FORMAT program.

2. An I/O (input/output) device handler program which interfaces with your computer to read and write data between the computer's memory and the devices attached to it. This program is on your DOS diskette in a file called IBMBIO.COM, but is not listed when you list the files on the diskette. It is put there by the FORMAT program and occupies a specific location on the diskette—therefore, if you want a copy of DOS on a diskette, you must format it first with the appropriate option before putting any other files on it.

3. A program called IBMDOS.COM, which also resides on your DOS diskette—like IBMBIO.COM, its filename does not appear when you list the files in the directory. The program contains a file manager and a series of service functions that can be used by any program which is designed to run under DOS's control. (All the programs on your DOS diskette are designed that way.)

How to start DOS

There are two ways to start DOS.

- With the power off, insert your DOS diskette in drive A and close the door. (It is always the built-in unit on the left.) Then turn the power on. The system automatically loads DOS into memory after 3 to 45 seconds of internal set-up, depending on the memory size.

- With the power on, insert the diskette in drive A and press the DEL key while holding down the CTRL and ALT keys. This sequence is called *System Reset.*

A few words about backups

It is strongly recommended that you make backup copies of all your diskettes, so that if a diskette somehow becomes damaged, or if files are accidentally erased, you will still have all your information. There are two ways to create a backup diskette (Remember to format it first.):

- Use the DISKCOPY command (See Chapter 2.) to create an exact image of an entire diskette on another diskette. You can use this command to copy the DOS diskette or your own diskette. The advantages of DISKCOPY are that it is the fastest way of copying a diskette, and that it copies everything, including DOS if it exists, in one operation.

  However, if either diskette has defective tracks, or if the diskette you want to copy has had a large amount of file creation/erasure activity, the method below is recommended because it compensates for the random placement of data caused by the creation/erasure activity.

- Use the COPY command (See Chapter 2.) to copy all files to the new diskette. This is slower, but it produces the same end result, with one difference– all your files will be written one right after the other (sequentially), which will "straighten out" the files, resulting in better performance if the diskette you are copying from had been subject to a lot of creation/erasure activity.

  If you use this method and want the new diskette to contain a copy of DOS, remember to tell the FORMAT command when you format the diskette because the system must reside on the diskette before any other files are placed on it.

  Note: In either case, the dates stored in the directory for each file are unaffected by copying.

Once DOS is loaded, it searches your DOS diskette for a file named COMMAND.COM and loads it into memory. The COMMAND program supplied with your DOS is a command processor that accepts commands you enter and runs the appropriate programs.

While COMMAND.COM is an important part of your system, it is distinguished from the programs described above because, unlike the above programs, you can replace it with your own if you wish. Because of the significant amount of function in the DOS command processor, however, replacing COMMAND.COM should only be considered by experienced programmers.

When the command processor is loaded (the one supplied with DOS), you will see this message on your screen.

```
Enter today's date (m-d-y):_
```

After a correct date is entered, DOS displays:

```
The IBM Personal Computer DOS
Version 1.00 (C)Copyright IBM Corp 1981
A>
```

The command processor is now ready to accept your input (the commands whose detailed descriptions are in Chapter 2).

Let's do it

Now, let's start up your system. Is your diskette in drive A? Yes? OK, now simply turn the system unit on and watch what happens. Be sure to enter a correct date separated by a / or - (such as 10-12-81), then press the ENTER key. (That's the one with ↵ on it.) Do the messages look like the ones described earlier? If not, repeat the process or use the System Reset described earlier.

You are now in control of a flexible combination of tools—your computer and your DOS.

Now what do I do?

Enter DIR on your keyboard, then press the ENTER key. The result is a display of the names, sizes and creation dates of the files on your DOS diskette. All of them with filename extensions of COM or EXE (such as BASIC.COM) are commands and are run by the system when you enter the command name and any required parameters.

Control-Function Keys

The following control-function keys are in effect when entering commands or input lines to any program Where two keys are specified (for example, CTRL-BREAK) press the second key while holding down the first key.

↵	This is the ENTER key, which causes the displayed line to be sent to the requesting program.
CTRL-BREAK	End the current operation.
CTRL-ENTER	Go to the next display line to continue entering the line being typed.
CTRL-NUMLOCK	Suspend system operation. Press any character key to resume the operation. This can be useful when a large amount of screen output is being generated. Press CTRL-NUMLOCK to temporarily halt output to the display so it can be read; press any other character key to restart.
CTRL-PRTSC	This key serves as an on-off switch for sending display output to the printer as well as the screen. Press these keys to send display output to the printer; press them again to **stop** sending display output to the printer. Although this allows the printer to function as a "system log", it slows down some operations because the computer waits during the printing. **Note:** This function is disabled when running either Disk or Advanced BASIC.

ESC	Cancel the current line and go to the next display line. A back slash (\) is displayed to indicate the cancelled line.
SHIFT-PRTSC	Send a copy of the current screen contents to the printer. This has the effect of printing a "snapshot" of the screen.
←	Backspace and remove the character from the screen. This is the key to the left of NUMLOCK, *not* key 4 in the numeric keypad.

DOS Editing Keys

DOS editing keys permit you to make corrections immediately to commands and input lines as they are being entered.

The DOS editing keys are used to edit **within** a line. The Line Editor program (See EDLIN in Chapter 3.) operates on **complete lines** within a file or document. When you are interacting with EDLIN and want to edit within a line, however, you use these DOS editing keys.

Any line entered from the keyboard is retained in an input buffer until ENTER is pressed. Then the line is made available to the interacting program for processing.

The line remains in the input buffer and can be thought of as a "template" for editing purposes. The DOS editing keys operate on this copy of the line. The line can be repeated or modified via the DOS editing keys, or an entirely new line can be entered.

This is a summary of the DOS editing keys and their functions.

DEL	Skip over one character in the retained line. (The cursor does not move.)
ESC	Cancel the line currently displayed. (The retained line remains unchanged.)
F1 or →	Copy one character from the retained line and display it.
F2	Copy all characters up to a specified character.
F3	Copy all remaining characters from the retained line to the screen.
F4	Skip over all characters up to a specified character (the opposite of F2).

F5 — Accept edited line for continued editing—the currently displayed line becomes the retained line, but it is not sent to the requesting program.

INS — Insert characters.

Using DOS Editing Keys

The following examples illustrate the use of the DOS editing keys while using the Line Editor (EDLIN) program. They can also be used, in the same way, to edit the last command entered or input to any standard DOS program. Assume that the full line in the template is "This is a sample file.".

> **Note:** Exceptions to the above occur when programs define special editing rules, such as used by some word processing programs. Another special case is the BASIC Program Editor used while programming in BASIC.

DEL — Skip over one character in the template (the opposite of the F1 key which copies one character). The cursor does not move.

Assume the screen looks like this:

```
1:*This is a sample file.
1:*_
```

If the DEL key is pressed twice and then F3 is pressed to copy the remainder of the line to the screen, the result shows that the first two characters are deleted.

```
1:*This is a sample file.
1:*
1:*is is a sample file._
```

ESC — Cancel the currently displayed line and go to the next display line. (The template remains unchanged.) A back slash (\) is displayed to indicate the displayed line is cancelled.

(The insert mode, if on, is turned off.)

Assume the screen looks like this:

```
1:*This is a sample file.
1:*_
```

If you want line 1 to have just the last two words, you can enter "Sample file".

```
1:*This is a sample file.
1:*Sample file_
```

But if you change your mind, press ESC to cancel the update.

```
1:*This is a sample file.
1:*Sample file\
   _
```

Now you can continue to edit the original line (This is a sample file.). Press F3 to copy the original line to the screen as shown below.

```
1:*This is a sample file.
1:*Sample file\
   This is a sample file._
```

F1 or → Copy one character from the template to the screen (the opposite of the DEL key which skips over one character in the template).

Assume the screen looks like this:

```
1:*This is a sample file.
1:*_
```

If the F1 or → key is pressed three times, the screen would look like:

```
1:*This is a sample file.
1:*Thi_
```

Each time the F1 or → key is pressed, one more character appears.

F2

F2 must be immediately followed by a single character. Copy all characters from the template to the screen, up to but not including the first occurrence of the specified character. (This function is similar to multiple use of the F1 key.)

If the specified character is not present in the template, nothing is copied.

Assume the screen looks like this:

```
1:*This is a sample file.
1:*_
```

If the F2 key is pressed, followed by a "p", the screen would look like:

```
1:*This is a sample file.
1:*This is a sam_
```

F3

Copy all the remaining characters in the template to the screen. (If you press ENTER instead, only the characters on the screen are sent to the requesting program.)

Assume the screen looks like this:

```
1:*This is a sample file.
1:*This is a sam_
```

If the F3 key is pressed, the screen would look like:

```
1:*This is a sample file.
1:*This is a sample file._
```

F4

F4 must be immediately followed by a single character. Skip over all characters in the template up to but not including the first occurrence of the specified character. (This function is similar to multiple use of the DEL key.) The cursor does not move.

If the specified character is not present in the template, no characters are skipped over.

Assume the screen looks like this:

```
1:*This is a sample file.
1:*_
```

If you press the F4 key, followed by a "p", and then press F3 to copy the remainder of the line, the screen would look like:

```
1:*This is a sample file.
1:*ple file._
```

F5

Put the currently displayed line into the template. (This is the same as pressing ENTER except the line is **not** sent to the requesting program.) An @ character is displayed to indicate the action. (The insert mode, if on, is turned off.)

You can now continue making changes to the line. When finished, press ENTER to send the line to the requesting program.

Assume the screen looks like this:

```
1:*This is a sample file.
1:*This is not a mailorder file._
```

If the replacement line is acceptable but still in need of editing, press F5 and the results would be:

```
1:*This is a sample file.
1:*This is not a mailorder file.@
   _
```

The second (replacement) line is now in the input buffer (template).

To remove the word "not" from the replacement line, you could first press F1 eight times:

```
1:*This is a sample file.
1:*This is not a mailorder file.@
   This is _
```

Then, press DEL four times and F3 once to copy the remaining characters.

```
1:*This is a sample file.
1:*This is not a mailorder file.@
   This is a mailorder file._
```

Now, press ENTER to make the replacement line (This is a mailorder file.) the template in place of the original line (This is a sample file.) and to send the line to the requesting program.

Or, press F5 again to put the displayed line into the template for further editing, without sending the line to the requesting program.

> **Note:** Pressing ENTER **immediately** after F5 empties the template (the input buffer).

INS

This key serves as an on/off switch for entering and leaving insert mode. Press the INS key to enter insert mode; press the INS key again to leave insert mode.

During the insert mode of operation, all characters entered are inserted in the line being displayed. The current position in the template is not changed.

When not in the insert mode of operation, all characters entered replace characters in the template. If at the end of a line, the characters are appended.

Assume the screen looks like this:

```
1:*This is a sample file.
1:*_
```

If the F2 key is pressed, followed by an "m", the screen would look like:

```
1:*This is a sample file.
1:*This is a sa_
```

To make an insertion, press the INS key and enter the characters "lary":

```
1:*This is a sample file.
1:*This is a salary_
```

The characters "lary" are inserted but no characters from the template are replaced.

Now, press INS again to leave insert mode and enter the four characters " tax".

```
1:*This is a sample file.
1:*This is a salary tax_
```

Now, press F3 to copy the remaining characters in the template to the display.

```
1:*This is a sample file.
1:*This is a salary tax file._
```

The "lary" was inserted but the "mple" was replaced by the " tax".

Press ENTER to make the replacement line (This is a salary tax file.) the template in place of the original line (This is a sample file.), and to send the line to the requesting program.

Or, press F5 again to put the displayed line into the template for further editing, without sending the line to the requesting program.

> **Note:** Pressing ENTER **immediately** after F5 empties the template (the input buffer).

DOS Commands

There are DOS commands that can be used to:

- compare, copy, display, erase, and rename files.
- compare, copy, and format diskettes.
- execute programs (system programs such as EDLIN and DEBUG, plus your own programs).
- analyze and list directories.
- enter date, time, and remarks.
- set various printer and screen options.
- transfer DOS to another diskette.
- request system wait.

This chapter explains how to use the DOS commands; the information appears in the following order:

- System startup and automatic program execution
- Summary of the commands supplied with DOS
- Definition of the DOS command parameters you supply
- Global filename characters
- Single-drive systems
- Common rules for all DOS commands
- Detailed description of the DOS commands.

System Startup

When you start up DOS (DOS diskette in drive A: and power ON, or restart by pressing System Reset) the DOS command processor is automatically loaded from the file named COMMAND.COM on your DOS diskette.

Automatic Program Execution

The command processor searches for a file named AUTOEXEC.BAT on the DOS diskette. This filename is special because it refers to a batch file (a file containing one or more DOS commands) that is automatically executed whenever the system is started. With this facility, you can cause programs or commands to be executed immediately every time the system is started. (See Batch Processing in this chapter for details on creating a batch file.)

If the AUTOEXEC.BAT file is found, it is immediately executed by the command processor.

Date Processing

If the AUTOEXEC.BAT file is **not** found you will see this prompt.

```
Enter today's date (m-d-y):_
```

where:

m is 1 or 2 digits from 1-12
d is 1 or 2 digits from 1-31
y is 2 digits from 80-99 (the 19 is assumed) or 4 digits from 1980-2099.

Any date is acceptable as today's date as long as the digits are in the correct ranges, and the delimiters between the numbers are either a slash (/) or hyphen (-).

If an invalid date or delimiter is entered, the system repeats the prompt.

Upon entering a proper date (for example, 10/12/81) you will see:

```
Enter today's date (m-d-y): 10/12/81
The IBM Personal Computer DOS
Version 1.00 (C)Copyright IBM Corp 1981
A>
```

The command processor is now ready to accept your commands. The date you entered will be recorded in the directory entry for any files that you create or alter.

Specifying the Default Drive

A> is the prompt from the command processor; whenever you see it, it means the system is waiting for you to enter a command.

The A is the default drive designation. The default drive is where DOS will look to find any filenames entered without a specified drive. You can change the default drive by entering the new designation letter followed by a colon. For example:

```
A>      (original prompt)
A>B:    (new drive designation)
B>      (new prompt)
```

Summary of DOS Commands

To enter a command, simply enter its name and any applicable parameters, then press ENTER.

There are two types of commands, internal (I) and external (E).

Internal commands (represented by I in the following table) are executed immediately because they reside in memory.

External commands (represented by E in the following table) reside on disk as program files; therefore, they must be read from disk before they can be executed. Any file with a filename extension of .COM or .EXE is considered to be an external command. This concept allows you to develop your own unique commands and add them to the system. (For example, programs such as FORMAT.COM and COMP.COM are external commands.) When entering an external command, do not include the filename extension.

Format Notation

The following notation is used in the format of commands.

- Words in capital letters are keywords. The specific characters shown must be entered, in any combination of upper/lower case.
- You are to supply any items in lower case italic letters.
- Items in square brackets ([ ]) are optional.
- An ellipsis (. . .) indicates an item may be repeated as many times as you wish.
- All punctuation (except square brackets) such as commas, equal signs, question marks, colons, or slashes must be included where shown.

Command	Type	Purpose	Format
(Batch)	I	Invoke batch files	[*d*:] *filename* [*parameters*]
CHKDSK	E	Check disk and report status	CHKDSK [*d*:]
COMP	E	Compare files	COMP [*filespec*] [*d*:] [*filename* [*.ext*]]
COPY	I	Copy files	COPY *filespec* [*d*:] [*filename* [*.ext*]]
DATE	E	Enter date	DATE
DIR	I	List filenames	DIR [*d*:] [*filename*[*.ext*]]
DISKCOMP	E	Compare diskettes	DISKCOMP [*d*:] [*d*:]
DISKCOPY	E	Copy diskettes	DISKCOPY [*d*:] [*d*:]
ERASE	I	Delete files	ERASE *filespec*
FORMAT	E	Format diskette	FORMAT [*d*:] [/S]
MODE	E	Set mode on printer/display	MODE [LPT#:] [,*n*] [,*m*] [,T]
PAUSE	I	System wait	PAUSE [*remark*]
REM	I	Display remark	REM [*remark*]
RENAME	I	Rename files	RENAME *filespec* *filename*[*.ext*]
SYS	E	Transfer DOS	SYS *d*:
TIME	E	Enter time	TIME
TYPE	I	Display file contents	TYPE *filespec*

DOS Command Parameters

Parameter	Definition
d:	Enter a drive letter followed by a colon to specify the drive (such as A: for the first drive on your system, B: for the second). If omitted, the **default** drive is assumed.
filename	Diskette filenames are 1-8 characters long, and may be immediately followed by a filename extension. Valid characters can be chosen from the following list. (An illegal character is assumed to be a delimiter; in that case, the filename would be truncated.)

A-Z	0-9	$	&	#	@	!
		%	'	(	)	-
		<	>	{	}	_
		\	^	~	¦	`

Several reserved names for specific devices can be used in place of a filename, and are listed below; they cannot be used as diskette filenames because the system assumes them to be the following devices. Any drive specifier or filename extension entered with these device names will be ignored. (The colon is optional.)

CON:	Console keyboard/screen. If used as an input device, the F6 key followed by the ENTER key can be used to generate an end-of-file indication to end its use as an input device.
AUX: or COM1:	First Asynchronous Communications Adapter port

LPT1:
or
PRN:
: Printer (as output device only)

NUL: Non-existent (dummy) device for testing applications. As an input device, immediate end-of-file is generated. As an output device, the write operations are simulated but no data is actually written.

.ext The optional filename extension consists of a period and 1-3 characters (same character set as for filename).

When referring to a file that has a filename extension, remember to include the extension; otherwise, the file will not be found.

filespec [d:]filename[.ext]

Examples:

```
B:myprog.COB
A:yourprog
DATAFILE.pas
cobfile
```

Global Filename Characters

Two special characters (? and *) can be used within a filename and its extension. These special characters give you greater flexibility with the DOS commands.

The ? Character

A ? in a filename, or filename extension, means that any character can occupy that position. For example,

```
DIR  AB?DE.XYZ
```

lists all directory entries on the default drive (No drive specifier was given.) with filenames that begin with AB, have any next character, followed by DE, and have an extension of XYZ. (The filename is five characters in length.)

The files listed by the DIR command might be:

```
ABCDE.XYZ
ABIDE.XYZ
ABODE.XYZ
```

The * Character

An * in a filename, or filename extension, means that any character can occupy that position, and all the remaining positions in the filename or extension. For example,

```
DIR  AB*.XYZ
```

lists all directory entries on the default drive with filenames that begin with AB and have an extension of XYZ. (In this case, the filename may be from 2-8 characters in length.)

The files listed by the DIR command might be:

```
ABCDE.XYZ
ABC357.XYZ
ABIDE.XYZ
ABIIOU.XYZ
ABO$$$.XYZ
ABODE.XYZ
```

Thus, the * character serves as a shorthand way of entering more than one ? character.

Some ? and * Examples

In each of the following examples, the DIR commands to drive A are equivalent.

Example 1. To list the directory entries for all files named INPUT, regardless of the filename extension, enter:

```
DIR A:INPUT.???
  or
DIR A:INPUT.*
```

Example 2. To list the directory entries for all files (regardless of their filename) with a filename extension of XYZ, enter:

```
DIR A:????????.XYZ
  or
DIR A:*.XYZ
```

Example 3. To list the directory entries for all files with filenames beginning with ABC and extensions beginning with E, enter:

```
DIR A:ABC?????.E??
  or
DIR A:ABC*.E*
```

Single-Drive Systems

In a single-drive system the commands are entered exactly the same as in a multi-drive system.

You should **think** of the single-drive system as having **two** drives (drive A and drive B). But, instead of A and B meaning two physical drives as in a multi-drive system, the A and B designate diskettes.

Therefore, if you specify drive B when the "drive A" diskette was last used, you are prompted to "switch drives" by changing diskettes. For example:

```
A>COPY COMMAND.COM B:
Insert diskette for drive B:
and strike any key when ready
    1 File(s) copied
A>_
```

If you specify drive A when the "drive B" diskette was last used, you are again prompted to change diskettes; this time for the diskette the system thinks of as "drive A".

The same procedure occurs if the command is executed from a batch file. The system waits for you to change diskettes and press a key before it continues.

> **Note:** Remember that the drive displayed in the system prompt (such as A>) is the drive where DOS looks to find a file whose name is entered without a driver specifier; it does not represent the last diskette (logical drive A or B) used.
>
> For example, assume that A: is the default drive. If the last operation performed was DIR B:, DOS believes the diskette for "drive B" is still in the drive. Issuing DIR A: or simply DIR (The **default** drive will be used.) causes DOS to prompt for the diskette for "drive A."

The DOS Commands

This section presents a detailed description of how to use the commands. The commands appear in alphabetical order; each with its format, type, purpose, and example where appropriate.

The following general information applies to all of the commands.

- Commands are usually followed by one or more parameters (except for DATE and TIME).
- Commands and parameters may be entered in upper or lower case, or a mixture of both cases.
- Commands and parameters **must** be separated by delimiters (space, comma, semicolon, equal sign, or the tab key). The delimiters can be mixed. For example:

```
COPY oldfile.rel;newfile.rel
RENAME,thisfile thatfile
```

- The three parts of *filespec* (d:filename.ext) must not be separated by delimiters. The (:) and (.) already serve as delimiters.
- Files are not required to have filename extensions when created or renamed. However, be sure to include the filename extension when referring to a file that **has** a filename extension.
- Commands can be ended while they are running by pressing CTRL-BREAK. Note that CTRL-BREAK is recognized only while the system is reading from the keyboard or printing characters on the screen. Thus, the command may not end immediately when you press CTRL-BREAK.
- Commands become effective only after the ENTER key is pressed.

- Global filename characters and device names are not allowed in a command name; they may be used only in command parameters.

- For commands producing a large amount of output, you can press CTRL-NUMLOCK to suspend the display (to read it before it scrolls away). Press any other key to restart the display.

- The control-function and DOS editing keys, described in Chapter 1, apply while entering the commands.

- The prompt from the command processing program is the default drive designation letter plus >, such as A>.

Batch Processing

Format: [*d*:] *filename* [*parameters*]

Type: Internal External
***

Purpose: From the designated or default drive, execute the commands contained in the specified file.

You must first create the file before you can cause it to be executed. All batch files must have a filename extension of .BAT.

You may pass parameters to the *filename*.BAT file when it is executed. Therefore, the file can do similar work during each execution, but with different data.

Notes:

1. Do not enter the name BATCH (unless the name of the file you want to execute is BATCH.BAT).

2. Only the *filename* must be entered. Do not give an extension.

3. The file named *filename*.BAT is executed.

4. If you press CTRL-BREAK while in batch mode, this prompt appears:

```
Terminate batch job (Y/N)?
```

   If you press Y, batch processing ends and the system prompt (such as A>) appears. If you press N, only the current command is ended and batch processing continues with the next command in the file.

5. If the diskette containing a batch file being processed is removed, DOS prompts you to insert it again before the next command can be read.

The AUTOEXEC.BAT File

A special case is the AUTOEXEC.BAT file. When you start or restart DOS, the command processor searches for the AUTOEXEC.BAT file. If present on the DOS diskette, it will be automatically executed whenever the system is started.

You can create a batch file using the Line Editor (EDLIN) or directly from the console (using COPY).

For example, to cause BASIC to be loaded and to run the program called MENU automatically, you would create an AUTOEXEC.BAT file as follows:

```
COPY CON: AUTOEXEC.BAT<ENTER>
```

Tells DOS to copy from the console (keyboard) into the AUTOEXEC.BAT file.

```
BASIC MENU<ENTER>
```

This statement goes into the AUTOEXEC.BAT file. It tells DOS to load BASIC and to run the MENU program whenever you start DOS.

Press the F6 key followed by the ENTER key to put the command BASIC MENU in the AUTOEXEC.BAT file. You are now set up to load BASIC and run the MENU program automatically whenever you start DOS.

To run your own BASIC program, enter the name of your program in place of MENU in the second line of the example.

Note: Remember that if AUTOEXEC is used, DOS does not prompt for the current date unless a DATE command is included in the AUTOEXEC.BAT file.

To Create a .BAT File with Replaceable Parameters

Within the batch file you may include "dummy" parameters, which will be replaced by values supplied when the batch file is executed.

For example:

```
A>Copy con: ASMFILE.BAT
Copy %1.MAC %2.MAC
Type %2.PRN
Type %0.BAT
<F6><ENTER>
1 File(s) copied
A>_
```

The file ASMFILE.BAT (which consists of three statements) now resides on the diskette in the default drive.

The dummy parameters (in this case %0, %1, and %2) are to be replaced sequentially by the parameters you supply when you invoke the file. (See below.) The dummy parameter %0 is always replaced by the drive designator (if specified) and filename of the batch file.

Notes:

1. Up to 10 dummy parameters (%0-%9) can be specified.

2. Because the dummy parameters use the percent sign, you must be careful about using the % character in a filename. If you want to use % as part of a filename **within** a batch file, you must specify it twice. For example, to specify the file ABC%.EXE you must enter it as ABC%%.EXE in the batch file.

To Execute a .BAT File with Replaceable Parameters

To execute ASMFILE.BAT and pass parameters, enter the batch filename followed by the parameters you want sequentially substituted for %1, %2, etc.

For example:

```
ASMFILE A:PROG1 B:PROG2
```

(ASMFILE is substituted for %0, A:PROG1 for %1, and B:PROG2 for %2.)

The result is the same as if you had entered each of the three commands (in the ASMFILE.BAT file) from the console with their parameters, as follows:

```
Copy A:PROG.1MAC B:PROG2.MAC
Type B:PROG2.PRN
Type ASMFILE.BAT
```

Remember that the dummy parameter %0 is always replaced by the drive designator (if specified) and filename of the batch file.

CHKDSK (Check Disk) Command

Format: CHKDSK [*d*:]

Type: Internal External
***

Purpose: Analyze the directory and file allocation table on the designated or default drive and produce a diskette and memory status report. This should be done occasionally for each diskette, to ensure the integrity of the file structures.

After checking the diskette, any error messages are displayed (See Appendix A.), then the following status report.

```
    XX disk files
XXXXXX bytes total disk space
XXXXXX bytes remain available

XXXXXXX bytes total memory
XXXXXXX bytes free
```

(The X's represent decimal integer values.)

Notes:

1. CHKDSK temporarily makes the drive specified in *d*: the default drive. Therefore, if CHKDSK is ended prematurely (for example, by requesting it be ended following a diskette error message), the default drive is changed to the drive that CHKDSK was checking.

2. CHKDSK assumes that the diskette to be checked is in the specified drive; there is no wait for the diskette to be inserted. Therefore, in a single-drive system, the specified drive should be different from the default drive unless the DOS diskette itself is being checked.

COMP (Compare Files) Command

Format: COMP [*filespec*] [*d:*] [*filename*[*.ext*]]

Type: Internal External
***

Purpose: Compare the contents of the first file to the contents of the second file. (This command compares two **files**; the DISKCOMP command compares two **entire diskettes**.)

The files may be on the same or different drives. Both filenames are optional; if omitted, you will be prompted for them. You are also prompted to insert the appropriate diskettes; COMP waits for you to press a key before beginning its compare.

COMP compares the files on a byte-for-byte basis. Unequal bytes result in messages giving the hexadecimal offset (into the file) of the unequal comparison and the bytes that were compared, as follows:

```
Compare error at offset XXXXXXXX
File 1 = XX
File 2 = XX
```

(where File 1 is the first filename entered and File 2 is the second).

After 10 unequal comparisons COMP assumes further comparing would be useless so processing ends with:

```
10 Mismatches - aborting compare
```

After a successful compare ("Files compare ok" message) or after a compare has ended, COMP displays:

```
Enter primary file name
Or strike the ENTER key to end_
```

COMP (Compare Files) Command

This offers the opportunity to compare two more files. To do so, enter the *filespec* of the first of the two files (COMP then prompts for the second). To end COMP processing, press only the ENTER key.

Notes:

1. The two files may have the same name, provided they are on different diskettes. (In this case, the drive specifiers must be different.)

2. If only a drive is specified for the second file, it is assumed that the second filename is the same as the first filename. In this case, too, the specified drives must be different.

3. Use of the special cnaracters (? and *) in the filenames do not cause multiple file compares. Only the first file matching each name is compared. (See "Global Filename Characters" earlier in this chapter.)

4. No compare takes place if the file sizes are different. You are prompted to compare other files.

COPY Command

Format: COPY *filespec* [*d*:] [*filename*[.*ext*]]

Type: Internal External
***

Purpose: This command:

- Copies one or more files (the required *filespec*) to another diskette, and gives the copies different names (the optional *filename.ext*) if you want to.
- Copies files to the same diskette. In this case you **must** give the copies different names. Otherwise, the copying is not permitted.

> **Note:** The special characters (? and *) can be used in the filename and extension parameters of both the original and duplicate files. See "Global Filename Characters" earlier in this chapter.

There are two basic format options with the COPY command.

Option 1

COPY *filespec*

or

COPY *filespec d*:

Copy a file to the default (first case) or designated (second case) drive. Because the name of the file is not changed, the from-drive (source drive) and to-drive (target drive) must be different.

For example, assume the default drive is A:

```
COPY B:MYPROG
```

copies the file MYPROG from the diskette in drive B, to the diskette in drive A (the default drive), with no change in the filename.

```
COPY *.* B:
```

copies all the files from the diskette in drive A (the default drive), to the diskette in drive B, with no change in the filenames or extensions. This is very useful if the files on the diskette in drive A were fragmented.

Option 2

COPY *filespec filename*[*.ext*]

or

COPY *filespec d:filename*[*.ext*]

Copy a file (*filespec*) to the default (first case) or designated (second case) drive, but change the name of the copy. Therefore, the from-drive (source drive) and to-drive (target drive) do not have to be different.

For example:

```
COPY MYPROG.ABC B:*.XXX
```

copies the file MYPROG.ABC from the diskette in drive A (the default drive), to the diskette in drive B, naming the copy MYPROG.XXX.

Note: The reserved device names can also be used for the copy operation. For example:

```
COPY CON: filespec
COPY CON: AUX:
COPY CON: LPT1:
COPY fileA CON:
COPY fileB AUX:
COPY fileC LPT1:
COPY AUX: fileX
COPY AUX: LPT1:
COPY AUX: CON:
```

Also, NUL: can be used in any variation.

DATE Command

Format: DATE

Type: Internal External
***

Purpose: Permits you to enter a date or change the date known to the system. This date is recorded in the directory entry for any files you create or alter.

You can change the date from the console or from a batch file. (Remember that the system does not prompt for the date if you use an AUTOEXEC.BAT file; so, you may want it to include a DATE command.)

The DATE command causes the following prompt.

```
Current date is mm-dd-yy
Enter new date:_
```

where:

m is 1 or 2 digits from 1-12
d is 1 or 2 digits from 1-31
y is 2 digits from 80-99 (the 19 is assumed)
or 4 digits from 1980-2099.

Notes:

1. If the date displayed is correct, simply press the ENTER key to leave it unchanged; you don't have to enter it again.

2. The valid delimiters within the date are the hyphen (-) and slash (/).

3. Any date is acceptable as today's date as long as the digits are in the correct ranges.

4. If an invalid date or delimiter is specified, you receive an "Invalid date" message.

DATE
Command

Example:

```
A>DATE
Current date is 07-17-82
Enter new date: 7/24/82_
```

DIR (Directory) Command

Format: DIR [*d*:] [*filename*[.*ext*]]

Type: Internal External
***

Purpose: List all directory entries or only those for specific files. The display for each file includes its size in bytes (decimal) and the date the file was last updated.

> **Note:** Directory entries for system files (IBMBIO.COM, IBMDOS.COM, and BADTRACK) are not listed, even if present.

There are two format options with the DIR command.

Option 1

DIR

or

DIR *d*:

List all directory entries from the default (first case) or designated (second case) drive. For example, the screen might look like:

```
FILE1    .A          100      11-17-81
FILE3    .A       104755      12-01-82
9x                  2600      02-28-92
```

DIR (Directory) Command

Option 2

DIR *filename.ext*

or

DIR *d:filename.ext*

List all directory entries from the default (first case) or designated (second case) drive that have the specified *filename.ext*.

> **Note:** The special characters (? and *) can be used in the filename and extension parameters. See "Global Filename Characters" earlier in this chapter.

Using the previous example, if you entered:

```
DIR FILE3.A
```

the screen would look like:

```
A>DIR FILE3.A
FILE3    .A    104755                    12-01-82
A>
```

Or if you entered:

```
DIR *.A
```

the screen would look like:

```
A>DIR *.A
FILE1    .A       100                    11-17-81
FILE3    .A    104755                    12-01-82
A>
```

DISKCOMP (Compare Diskette) Command

Format: DISKCOMP [*d*:] [*d*:]

Type: Internal External
***

Purpose: Compare the contents of the diskette in the first drive (primary) to the contents of the diskette in the second drive (secondary). Normally, this would be done following a DISKCOPY operation. (This command compares **entire diskettes**; the COMP command compares two **files**.)

The drives can be different, or the same (resulting in a single-drive compare operation). You are prompted to insert the diskettes at the appropriate times; DISKCOMP waits for you to press any key before continuing.

DISKCOMP compares all 40 tracks on a track-for-track basis and issues a message indicating the track number (0-39) when they are not equal.

After completing the compare, DISKCOMP prompts:

```
Compare more diskettes? (Y/N)_
```

To end the command, press N. It you press Y, the next compare is done on the same drives after prompts to insert the proper diskettes.

DISKCOMP (Compare Diskette) Command

Notes:

1. If both parameters are omitted, a single-drive compare operation is performed on the default drive.

2. If the second parameter is omitted, the default drive is used as the secondary. (If the default drive was specified as the primary, then this also results in a single-drive compare operation.)

3. DISKCOMP attempts to keep the number of diskette insertions to a minimum by reading the maximum amount of data that memory can hold before comparing it, and by reversing the order of disk reads each time. Depending on the amount of available memory, this can result in 2, 4, 8, or 10 tracks of data being read for each diskette insertion.

4. DISKCOMP will not normally give a "Diskettes compare OK" message if you try to compare a backup diskette created by the COPY command. The COPY operation condenses file space when it can, thus producing a copy that contains the same information but in a different physical sequence. In this case, use the COMP command to compare individual files on the diskettes.

5. In a single-drive system, all prompts will be for drive A, regardless of any drive specifiers entered.

DISKCOPY (Copy Diskette) Command

Format: DISKCOPY [*d*:] [*d*:]

Type: Internal External
***

Purpose: Copy the contents of the source diskette ('from' diskette) to the target diskette ('to' diskette) in the second drive.

The drives can be different, or the same (resulting in a single-drive copy operation). You are prompted to insert the diskettes at the appropriate times; DISKCOPY waits for you to press any key before continuing.

After copying, DISKCOPY prompts:

```
Copy another? (Y/N)_
```

To end the command, press N. If you press Y, the next copy is done on the same drives after prompts to insert the proper diskettes.

Notes:

1. If both parameters are omitted, a single-drive copy operation is performed on the default drive.

2. If the second parameter is omitted, the default drive is used as the 'target' drive. (If the default drive was specified as the 'source' drive, then this also results in a single-drive copy operation.)

3. DISKCOPY attempts to keep the number of diskette insertions to a minimum by reading the maximum amount of data that memory can hold before writing it to the 'target' diskette. Depending on the amount of available memory, this can result in 2, 4, 8, or 10 tracks of data being processed for each pair of diskette insertions.

4. Diskettes with much file creation and deletion activity become fragmented because disk space is not allocated sequentially. (The first free sector found is the next sector allocated, regardless of its location on the diskette.)

   A fragmented disk can cause degraded performance due to excessive head movement and rotational delays involved in finding, reading, or writing a file.

   If this is the case, it is recommended that you use the COPY command, instead of DISKCOPY, to eliminate the fragmentation.

   For example:

   ```
   COPY A:*.* B:
   ```

   copies all the files from the diskette in drive A (regardless of their fragmentation), to the diskette in drive B in **sequential** sectors.

5. DISKCOMP is recommended to verify a successful DISKCOPY.

6. In a single-drive system, all prompts will be for drive A, regardless of any drive specifiers entered.

ERASE Command

Format: ERASE *filespec*

Type: Internal External
***

Purpose: Delete the file with the specified filename from the designated drive (or default if no drive was specified).

Notes:

1. The special characters (? and *) can be used in the filename and extension. See "Global Filename Characters" earlier in this chapter. This should be done with caution, however, because multiple files can be erased with a single command.

2. To erase all files on a diskette, enter:

```
ERASE [d:]*.*
```

3. The system files IBMBIO.COM, IBMDOS.COM, and BADTRACK cannot be erased.

Example:

```
A>ERASE A:myprog.1
```

FORMAT Command

Format: FORMAT [*d*;] [/S]

Type: Internal External
***

Purpose: This command:

- initializes the diskette in the designated (or default) drive to a recording format acceptable to DOS,
- analyzes the entire diskette for any defective tracks, and
- prepares the diskette to accept DOS files by initializing the directory, file allocation table, and system loader.

This must be done for all new diskettes before they can be used by DOS.

If /S is specified, the operating system files are also copied from the default drive to the diskette in the following order.

IBMBIO.COM
IBMDOS.COM
COMMAND.COM

Notes:

1. The formatting process destroys any previously existing data on the diskette.
2. During the formatting process, any defective tracks are allocated to a file called BADTRACK. This prevents them from being allocated to a data file.

3. Directory entries for IBMBIO.COM, IBMDOS.COM, and BADTRACK will not appear in any directory searches (including the DIR command).

4. To determine whether there are any defective tracks, analyze the status report displayed by the CHKDSK command.

   If /S was not specified during formatting, CHKDSK should report 0 disk files. If 1 disk file is reported, that means defective tracks were found and the file being reported is BADTRACK.

   If /S was specified during formatting, CHKDSK should report 3 files present; a report of 4 files means defective tracks were found.

Example:

```
A>FORMAT B:/S
Insert new diskette for drive B:
and strike any key when ready
Formatting...Format complete
System transferred
Format another (Y/N)? _
```

Press Y to format another diskette or N to end the FORMAT program.

MODE Command

Format: MODE [LPT#:] [,*n*] [,*m*] [,T]

Type: Internal External
***

Purpose: Set the mode of operation on a printer, or on a display connected to the Color/Graphics Monitor Adapter. (This command has no effect on a display connected to the IBM Monochrome Display and Parallel Printer Adapter.)

A missing or invalid *n* or *m* parameter means the mode of operation for that parameter is not changed.

There are two options with the MODE command.

Option 1 (For printer)

MODE LPT#:[,*n*] [,*m*]

where:

is 1, 2, or 3 (printer number)
n is 80 or 132 (characters per line)
m is 6 or 8 (lines per inch vertical spacing)

For example:

```
MODE LPT1:,132,8
```

sets the mode of operation of printer 1 to 132 characters per line and 8 lines per inch vertical spacing.

> **Note:** The power-on default options are 80 characters per line and 6 lines per inch.

Option 2 (For display connected to Color/Graphics Monitor Adapter)

MODE [*n*] [,*m*] [,T]

where:

n is 40 or 80 (characters per line)
m is R or L (shift display right or left)
T requests a test pattern useful to align the display.

For readability the display can be shifted one character (for 40-column) or two characters (for 80-column) in either direction. If T is specified, a prompt will ask if the screen is aligned properly. If you enter Y the command ends. If you enter N the shift is repeated followed by the same prompt. For example,

```
MODE 80,R,T
```

sets the mode of operation to 80 characters per line and shifts the display two character positions to the right. The test pattern is displayed to give you the opportunity to further shift the display without having to enter the command again.

PAUSE Command

Format: PAUSE [*remark*]

Type: Internal External
***

Purpose: The system waits after displaying the optional *remark* and the message "Strike a key when ready. . . ." The optional remark can be any string of characters up to 121 bytes long.

Inserting PAUSE commands within a batch file can be useful to display messages and allow you to change diskettes between commands. Press any key except CTRL-BREAK to resume execution of the batch file.

You can control how much of the batch file is executed by placing PAUSE commands at strategic points in the file. At each PAUSE command the system will stop, giving you time to decide whether to end the processing (Press CTRL-BREAK.) or continue processing. (Press any other key.)

Example:
```
A>PAUSE Change diskette in drive A
Strike a key when ready..._
```

REM (Remark) Command

Format: REM [*remark*]

Type: Internal External
***

Purpose: Display remarks from within a batch file. They are displayed when the batch execution reaches the remark.

The remark can be any string of characters up to 123 bytes long.

REM commands without remarks can be used for spacing within your batch file, for readability.

Example:
```
REM This is the daily checkout program.
```

RENAME Command

Format: RENAME *filespec filename*[*.ext*]

Type: Internal External
***

Purpose: Change the name of the file specified in the first parameter to the name and extension given in the second parameter. (A valid drive specified in the second parameter is ignored.)

> **Note:** The special characters (? and *) can be used in the parameters. See "Global Filename Characters" earlier in this chapter.

Example:

```
RENAME B:ABODE ?D?B?
```

renames the file ABODE on drive B to ADOBE.

```
RENAME B:ABODE *.XY
```

renames the file ABODE on drive B to ABODE.XY.

SYS (System) Command

Format: SYS *d*:

Type: Internal External
***

Purpose: Transfer the operating system files from the default drive to the specified drive, in the following order.

IBMBIO.COM
IBMDOS.COM

The diskette in the specified drive must have been previously formatted by a FORMAT d:/S command to contain a copy of DOS. If not, the system cannot be transferred because the specific diskette locations required for the system files have not been allocated for it.

> **Note:** SYS is provided so that you may transfer a copy of DOS to an application program diskette designed to use DOS, but sold without it.

TIME Command

Format: TIME

Type: Internal External
***

Purpose: Permits you to enter or change the time known to the system. You can change the time from the console or from a batch file.

The TIME command causes the following prompt.

```
Current time is hh:mm:ss.xx
Enter new time:_
```

where:

hh is 1 or 2 digits from 0-23 (hours)
mm is 1 or 2 digits from 0-59 (minutes)
ss is 1 or 2 digits from 0-59 (seconds)
xx is 1 or 2 digits from 0-99 (hundredths of seconds)

Notes:

1. To leave the time as is, simply press ENTER.

2. If you enter any information, such as the hours, then press ENTER, the remaining fields are set to zero.

3. Any time is acceptable as long as the digits are in the correct ranges.

4. The valid delimiters within the time are the colons (:) separating the hours, minutes, and seconds; and the period (.) between the seconds and the hundredths of seconds.

5. If an invalid time or delimiter is specified, you receive an "Invalid time" message.

Example:

```
A>TIME
Current time is 00:25:16.65
Enter new time: 13:55:00.00_
```

TYPE Command

Format: TYPE *filespec*

Type: Internal External
***

Purpose: Display the contents of the specified file on the screen.

The data is unformatted except that tab characters are expanded to an 8-character boundary; that is, columns 8, 16, 24, etc.

Notes:

1. By pressing CTRL-PRTSC, you can cause the file contents to be printed as it is being displayed.

2. Text files may appear in a legible format, but others such as object program files may appear unreadable due to the presence of non-alphabetic or non-numeric characters.

Example:

```
TYPE B:myprog.one
```

The Line Editor (EDLIN)

The Line Editor (EDLIN) can be used to create, alter, and display **source files** (unassembled programs in source language format) or text files.

EDLIN is a line text editor which can be used to:

- create new source files and save them
- update old files and save both updated and original files
- delete, edit, insert, and display lines
- search for, delete, or replace text within one or more lines.

The text of files created or edited by EDLIN is divided into lines of varying length, up to 253 characters per line.

Line numbers are dynamically generated and displayed by EDLIN during the editing process, but are not actually present in the saved file. When inserting lines, all line numbers following the inserted text are advanced automatically by the number of lines inserted. When deleting lines, all line numbers following the deleted text are decreased automatically by the number of lines deleted. Consequently, line numbers always go consecutively from 1 through n (the last line).

This chapter explains how to use the EDLIN program The information appears in the following order.

- How to invoke the EDLIN program
- Summary of the commands you can use with EDLIN
- Definition of the EDLIN command parameters you supply
- Detailed description of the EDLIN commands

How to Invoke the EDLIN Program

To invoke EDLIN, enter:

EDLIN *filespec*

- If the specified file exists on the designated drive (or default if no drive was specified), the file is loaded until memory is 75% full. If the entire file is loaded, a message and prompt (*) are displayed as follows:

```
End of input file
*_
```

  You can now edit the file.

- If the entire file cannot be loaded, EDLIN loads lines until memory is 75% full, then displays the prompt (*). You can now edit the portion of the file in memory.

  To edit the rest of the file, edited lines in memory must be written to diskette in order to free memory for unedited lines to be loaded from diskette. (See the Write Lines and Append Lines commands.)

- If the specified file does not exist on the designated drive (or default if no drive was specified), a new file is created with the specified name. A message and prompt (*) are displayed as follows:

```
New file
*_
```

  You can now create a new file by entering the desired lines of text (first enter an I command for Insert Lines).

When the editing session ends, the original and updated (or new) files can be saved. (See the End Edit command.) The original file is renamed to an extension of .BAK, and the new file has the filename and extension specified on the EDLIN command.

Notes:

1. A file with a filename extension of .BAK cannot be edited with EDLIN because it is assumed to be a backup file. If it is necessary to edit such a file, RENAME it to another extension and then invoke EDLIN specifying the new name.

2. When invoked, EDLIN first erases the backup copy (.BAK) of the file if it exists, to ensure adequate space on the diskette for the updated file. It then allocates a new file with the filename specified on the EDLIN command and an extension of .$$$. This is used to contain the updated file, and is ultimately renamed to the specified *filespec* by the End Edit command.

Summary of EDLIN Commands

See "Format Notation" in Chapter 2 for an explanation of the notation used in the format of the following commands.

Command	Format
Append Lines	[*n*] A
Delete Lines	[*line*] [,*line*] D
Edit Line	[*line*]
End Edit	E
Insert Lines	[*line*] I
List Lines	[*line*] [,*line*] L
Quit Edit	Q
Replace Text	[*line*] [,*line*] [?] R*string*[<F6>*string*]
Search Text	[*line*] [,*line*] [?] S*string*
Write Lines	[*n*] W

The EDLIN Command Parameters

Parameter	Definition
line	Three possible entries can be made. • Enter a decimal integer from 1-65529. Specifying a number greater than the number of lines in memory indicates the line after the last line that exists (same effect as #, see below). Line numbers must be separated from each other by a comma or space. OR • Enter a period (.) to specify the current line. The current line is: – Used to indicate the location of the last change to the file. – Not necessarily the last line displayed. – Marked by an asterisk (*) between the line number and the first character of text in the line. 10*FIRST CHARACTER OF TEXT OR • Enter a pound sign (#) to specify the line after the last line in memory. This has the same effect as specifying a number greater than the number of lines in memory.
n	Enter the number of lines to be written to diskette or loaded from diskette (and appended to the file in memory).

This parameter is used only with the Write Lines and Append Lines commands. These commands are meaningful only if the file to be edited is too large to fit in memory.

string

Enter one or more characters to represent text to be found, replaced, deleted, or to replace other text.

This parameter is used only with the Search Text and Replace Text commands.

The EDLIN Commands

This section presents a detailed description of how to use the commands to EDLIN. The commands appear in alphabetical order; each with its format, function, and example where appropriate.

The following general information applies to the EDLIN commands.

- A command is a single letter (except the Edit Line command).
- Commands are usually preceded and/or followed by parameters (except the End Edit and Quit Edit commands).
- Commands and *string* parameters can be entered in upper or lower case; or a mixture of both cases.
- Commands and parameters may be separated by delimiters (spaces or commas) for readability, but a delimiter is required only between two adjacent line numbers.
- Commands become effective only after the ENTER key is pressed.
- Commands can be ended by pressing CTRL-BREAK.
- For commands producing a large amount of output, press CTRL-NUMLOCK to suspend the display to read it before it scrolls away. Press any other character to restart the display.
- The control-function and DOS editing keys, described in Chapter 1, apply while using EDLIN. They are very useful for editing within a line while the EDLIN commands are intended for editing operations on entire lines.
- The prompt from EDLIN is an asterisk (*).

Append Lines Command

Format: [*n*] A

Purpose: Append *n* lines from diskette to the file in memory being edited. The lines are added to the end of the current lines in memory.

This command is meaningful only if the file being edited was too large to fit in memory. As much as possible was read into memory for editing.

To edit the rest of the file, edited lines in memory must be written to diskette (See the Write Lines command.) before unedited lines can be read from diskette via the Append Lines command.

Notes:

1. If the number of lines is not specified, lines are appended until available memory is 75% full. (No action is taken if available memory is already at least 75% full.)

2. The message "End of input file" is displayed when the Append Lines command (A) has read the last line of the file into memory.

Delete Lines Command

Format: [*line*] [,*line*] D

Purpose: Delete the specified range of lines.

The line following the deleted range becomes the current line (even if the deleted range includes the last line in memory). The current line and all the following lines are renumbered.

Defaults occur if either one or both of the parameters are omitted.

If the first parameter is omitted,

,*line* D

deletion starts with the current line and ends with the line specified by the second parameter. (The beginning comma is required to indicate the omitted first parameter.)

If the second parameter is omitted,

line D

or

line, D

only the single specified line is deleted.

If both parameters are omitted,

D

only the current line is deleted, and the line which follows becomes the current line.

Delete Lines Command

Example: Assume that the following file is ready to edit. Line 29 is the current line.

```
 1: This is a sample file
 2: used to demonstrate
 3: line deletion
 4: and dynamic
 5: line number generation.
 .
 .
 .
25: See what happens
26: to the lines
27: and line numbers
28: when lines are
29:*deleted.
```

To delete a range of lines, from 5-25, enter:

```
5,25 D
```

The result would be:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
4: and dynamic
5:*to the lines
6: and line numbers
7: when lines are
8: deleted.
```

Lines 5-25 are deleted from the file, lines 26-29 are renumbered to 5-8, and line 5 becomes the current line.

To delete the current and the following line, enter:

```
,6 D
```

The result would be:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
4: and dynamic
5:*when lines are
6: deleted.
```

Lines 5-6 are deleted from the file and lines 7-8 are renumbered to 5-6. Line 5 is still the current line but now has different text.

To delete a single line, say line 2, enter:

```
2 D
```

The result would be:

```
1: This is a sample file
2:*line deletion
3: and dynamic
4: when lines are
5: deleted.
```

Line 2 is deleted and lines 3-6 are renumbered to 2-5. The new line 2 becomes the current line.

To delete only the current line, enter:

```
D
```

The result would be:

```
1: This is a sample file
2:*and dynamic
3: when lines are
4: deleted.
```

The current line (#2) was deleted and lines 3-5 are renumbered to 2-4. The new line 2 becomes the current line.

Edit Line Command

Format: [*line*]

Purpose: Enter the *line* number of the line to be edited (or enter . to indicate the current line).

Making no entry (by pressing ENTER only) specifies that the line after the current line is to be edited.

The line number and its text are displayed, then the line number is repeated on the line below.

The control-function and editing keys, described in Chapter 1, may be used to edit the line, or the line may be completely replaced by simply typing new text.

The edited line is placed in the file when the ENTER key is pressed, and becomes the current line.

If you decide not to save the changed line, press either ESC or CTRL-BREAK. The original line remains unchanged. (Pressing ENTER with the cursor at the beginning of the line has the same effect.)

Remember that if the cursor is in any position other than the beginning or end of the line, pressing ENTER truncates the rest of the line.

Example:

```
*6
   6: This is a sample unedited line.
   6:
```

The first line above is your request to edit line 6, followed by the 2-line display response.

To move the cursor to the letter "u", enter:

```
<F2 >u
```

The result would be:

```
*6
   6: This is a sample unedited line.
   6: This is a sample
```

To delete the next two characters and then include the rest of the line, enter:

```
<DEL><DEL><F3>
```

The result would be:

```
*6
   6: This is a sample unedited line.
   6: This is a sample edited line.
```

Now you may take one of four actions:

- Press ENTER to save the changed line.
- Extend the changed line by typing more text. (You are automatically in insert mode when the cursor is at the end of a line.)
- Press F5 to do more editing with the changed line without changing the original line.
- Press ESC or CTRL-BREAK to cancel the changed line. (The original contents of the line are preserved.)

End Edit Command

Format: E

Purpose: End EDLIN and save the updated file.

The edited file is saved by writing it to the drive and filename specified when EDLIN was invoked.

The original file (the one specified when EDLIN was invoked) is renamed by giving it a filename extension (.BAK). No .BAK file is created if there was no original file (that is, if the editing session created a new file rather than updating an old file).

EDLIN returns to the DOS command processor which issues the normal command prompt.

> **Note:** Be certain the diskette has enough free space to save the entire file. If not, only a portion of the file is saved; the portion in memory not written to diskette is lost.

Insert Lines Command

Format: [*line*] I

Purpose: Insert lines of text immediately **before** the specified *line*.

If *line* is not specified (or is specified as .) the insertion is made immediately before the current line.

If *line* is greater than the highest existing line number (or is specified as #) the insertion is made after the last line in memory.

EDLIN displays the appropriate line number so you can enter one or more lines, ending each by pressing ENTER. During the insert mode of operation, successive line numbers appear automatically each time ENTER is pressed.

Press CTRL-BREAK to leave the insert mode of operation.

The line that follows the inserted lines becomes the current line (even if the inserted lines were added to the end of the lines in memory). The current line and all the following lines are renumbered.

> **Note:** When creating a new file, the Insert Lines command (I) must be entered before text can be inserted.

Insert Lines Command

Example: Assume that the following file is ready to edit and line 3 is the current line.

```
1: This is a sample file
2: used to demonstrate
3:*line deletion
4: and dynamic
5: line number generation
```

To insert text before line 4, the entry and immediate response would look like:

```
4 I
  4:*_
```

Now, to insert two new lines of text:

```
*4 I
   4: First new line of text
   5: Second new line of text
   6: <CTRL-BREAK>
```

(The original lines 4 and 5 are now renumbered to 6 and 7.)

Displaying the file now with a List Lines command would look like:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
4: First new line of text
5: Second new line of text
6:*and dynamic
7: line number generation.
```

If the two inserted lines had been placed at the beginning of the file (1 I), the screen would have looked like:

```
1: First new line of text
2: Second new line of text
3:*This is a sample file
4: used to demonstrate
5: line deletion
6: and dynamic
7: line number generation.
```

If the two inserted lines had been placed immediately before the current line (3 I or . I or I), the screen would have looked like:

```
1: This is a sample file
2: used to demonstrate
3: First new line of text
4: Second new line of text
5:*line deletion
6: and dynamic
7: line number generation.
```

If the two inserted lines had been placed at the end of the file (6 I or # I), the screen would have looked like:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
4: and dynamic
5: line number generation.
6: First new line of text
7: Second new line of text
```

List Lines Command

Format: [*line*] [,*line*] L

Purpose: Display the specified range of lines.

The current line remains unchanged.

Defaults occur if either one or both of the parameters are omitted.

- If the first parameter is omitted (The beginning comma is required to indicate the omitted first parameter.)

  ,*line* L

  the display starts 11 lines before the current line and ends with the specified line.

  > **Note:** If the specified line is more than 11 lines before the current line, the display is the same as if both parameters were omitted. (See below.)

- If the second parameter is omitted,

  line L

  or

  line, L

  a total of 23 lines is displayed, starting with the specified line.

- If both parameters are omitted,

  L

  a total of 23 lines is displayed (the 11 lines before the current line, the current line, and the 11 lines after the current line). If there aren't 11 lines before the current line, then extra lines (from after the current line) are displayed to make a total of 23 lines.

Example: Assume that the following file is ready to edit and line 15 is the current line.

```
 1: This is a sample file
 2: used to demonstrate
 3: line deletion
 4: and dynamic
 5: line number generation.
 .
 .
 .
15:*This is the current line (note the asterisk)
 .
 .
 .
25: See what happens
26: to the lines
27: and line numbers
28: when lines are
29: deleted.
```

To display a range of lines, from 5-25, enter:

```
5,25 L
```

The screen would be:

```
 5: line number generation.
 .
 .
 .
15:*This is the current line (note the asterisk)
 .
 .
 .
25: See what happens
```

To display the first 3 lines, enter:

```
1,3 L
```

The screen would be:

```
1: This is a sample file
2: used to demonstrate
3: line deletion
```

To display 23 lines, starting with line 3, enter:

```
3 L
```

The screen would be:

```
 3: line deletion
 4: and dynamic
 5: line number generation.
 .
 .
 .
15:*This is the current line (note the asterisk)
 .
 .
 .
25: See what happens
```

To display 23 lines centered around the current line, enter:

```
L
```

The screen would be:

```
 4: and dynamic
 5: line number generation
  .
  .
  .
15:*This is the current line (Note the asterisk.)
  .
  .
  .
25: See what happens
26: to the lines
```

Quit Edit Command

Format: Q

Purpose: Quit the editing session without saving changes. EDLIN prompts to be sure you really don't want to save the changes.

Example:

```
Q
Abort edit (Y/N)?_
```

Enter Y to quit the editing session. No editing changes are saved and no .BAK file is created. (See the End Edit command.) If you really don't want to quit, enter N or any other character to continue the editing session.

> **Note:** When invoked, EDLIN erases any previous backup copy of the file (filename.BAK) to make room for saving a new copy. Therefore, if you reply Y to the above prompt, your previous backup copy will no longer exist.

Replace Text Command

Format: [*line*] [,*line*] [?] R*string*[<F6>*string*]

Purpose: Replace all occurrences of the first *string* in the specified range of lines with the second *string*. If the second *string* is omitted, delete all occurrences of the first *string* within the specified range of lines.

Display every changed line each time it is modified. The last line changed becomes the current line.

Specify the optional parameter (?) to request a prompt (O.K.?) after each display of a modified line. Press the Y or ENTER key to accept the modification. Enter any other character to reject the modification. In either case, the search continues for further occurrences of the first *string* within the range of lines, including multiple occurrences within the same line.

Because omitting the first *line* defaults to line 1, and omitting the second *line* defaults to the last line, omitting both *line* parameters means to search all the lines in memory for occurrences of the first *string*.

> **Note:** The first *string* begins with the character in the position immediately following the R, and continues until ended by <F6> or <CTRL-Z> (or by the ENTER key if the second *string* is omitted). If the first *string* is omitted, no search can be made so the command ends immediately with a "Not found" message.
>
> The second *string* begins immediately following the <F6> or <CTRL-Z> and continues until ended by the ENTER key.

Replace Text Command

Example: Assume that the following file is ready to edit and line 7 is the current line.

```
1: This is a sample file
2: used to demonstrate
3: the Replace and Search Text commands.
4: This includes the
5: optional parameter ?
6: and required string
7:*parameter.
```

To replace all occurrences of "and" with "or" in the file, enter:

```
1,7 Rand<F6>or<ENTER>
```

The result would be:

```
3: The Replace or Search Text commors.
6: or required string
```

Line 6 becomes the current line in the file. Lines 1, 2, 4, 5, and 7 are not displayed because they were not changed.

Greater selectivity can be achieved by requesting a prompt (by means of the ? parameter) after each display of a modified line. In this case, the screen would look like:

```
*1,7?Rand<F6>or<ENTER>
     3: the Replace or Search Text commands.
O.K.? Y
     3: the Replace or Search Text commors.
O.K.? N
     6: or required string
O.K.? Y
*
```

Lines 3 and 6 would look like:

```
3: the Replace or Search Text commands.
6: or required string
```

Search Text Command

Format: [*line*] [,*line*] [?] S*string*

Purpose: Search the specified range of lines for a match of the specified *string*.

The first line to contain a match is displayed and the search ends, unless the ? parameter is used. The first matching line found and selected becomes the current line.

If no match is found the search ends after displaying "Not found". The current line remains unchanged.

Specify the optional parameter (?) to request a prompt (O.K.?) after each display of a line containing a match. Press the Y or ENTER key to cause the matching line to become the current line and the search to end. Enter any other character to continue the search until another match is found, or until all lines within the range have been searched (then the "Not found" is displayed).

Because omitting the first *line* defaults to line 1, and omitting the second *line* defaults to the last line, omitting both *line* parameters means to search all the lines in memory.

> **Note:** The *string* begins with the character in the position immediately following the S, and continues until ended by pressing the ENTER key. If *string* is omitted, no search can be made so the command ends immediately with a "Not found" message.

Search Text Command

Example: Assume that the following file is ready to edit and line 7 is the current line.

```
1: This is a sample file
2: used to demonstrate
3: the Search Text command.
4: This includes the
5: optional parameter ?
6: and required string
7:*parameter.
```

To search for the first occurrence of "and" in the file, enter:

```
1,7 Sand
  or
1,  Sand
  or
,7  Sand
  or
Sand
```

The result would be:

```
     3: the Search Text command.
*
```

The "and" is part of the word "command". Line 3 becomes the current line in the file.

Perhaps this is not the "and" you are looking for. To continue the search, enter a new starting line number (4), press F3 to copy the remainder of the previous Search Text command (S) to the screen, and press ENTER to execute the S command starting at line 4.

The screen would look like:

```
*1,7 Sand
     3: the Search Text command.
*4,7 Sand
     6: and required string
*
```

Line 6 now becomes the current line in the file.

You can also do this by requesting a prompt (by means of the ? parameter) after each display of a matching line. In this case, the screen would look like:

```
*1,7 ? Sand
     3: the Search Text command.
O.K.? N
     6: and required string
O.K.? Y
*
```

Write Lines Command

Format: [*n*] W

Purpose: Write *n* lines to diskette from the lines in memory being edited. Lines are written beginning with line number 1.

This command is meaningful only if the file being edited is too large to fit in memory. In this case, lines were read until memory was 75% full.

To edit the rest of the file, edited lines in memory must be written to diskette (via this command) before additional lines can be read from diskette via the Append Lines command.

> **Note:** If the number of lines is not specified, lines are written until available memory is only 25% used. (No action is taken if available memory is already less than 25% used.) All lines are renumbered so that the first remaining line becomes number 1.

The Linker (LINK) Program

LINK is a program that:

- Combines separately produced object modules.
- Searches library files for definitions of unresolved external references.
- Resolves external cross references.
- Produces a printable listing that shows the resolution of external references and error messages.
- Produces a relocatable load module.

Files

The linker processes the following input, output, and temporary files.

Input Files

Type	.ext		Produced by
	Default	Override	
Object	.REL	Yes	Assembler
Object	.OBJ	Yes	Compiler
Library	(none)	(none)	Compiler
Automatic Response	(none)	(none)	User

Output Files

Type	.ext		Used by
	Default	Override	
Listing	.MAP	Yes	User
Run	.EXE	No	Relocatable loader (COMMAND.COM)

VM.TMP (Temporary File)

LINK makes use of as much memory as is available to hold data defining the load module being created. If the resultant module is too large to be processed with the available amount of memory, the linker may need additional memory space. In this case, a temporary file called VM.TMP is created on the diskette in the DOS default drive. A message is displayed to indicate when this overflow to diskette has begun.

Once such a file is created, that diskette should not be removed until LINK ends. When LINK ends, the VM.TMP file is deleted.

If the DOS default drive already has a file by the name of VM.TMP, it will be deleted by LINK and a new one reallocated. The contents of the previous file are destroyed. You must not use VM.TMP as one of your own filenames.

Command Prompts

After you have started the linker session, you respond to a series of nine prompts. These prompts may be responded to individually from the keyboard or automatically via a special diskette file. (See Linker Invocation.)

LINK prompts you for the names of object, run, list, and library files; and for optional parameters that govern the linker session. When the session is finished, LINK returns to DOS. The DOS prompt is displayed when LINK has finished. If the LINK is unsuccessful, LINK returns a message.

The prompts are described below in their order of appearance. The default is shown in square brackets ([ ]), in the Responses column, following the prompt. Prompts not followed by a default require a response from you.

Prompt	Responses
OBJECT MODULES:	*filespec*[*filespec*. . .]
RUN FILE:	*filespec* [/P]
LIST FILE:	[*filespec*]
LIBRARIES:	*filespec*[*filespec*. . .]
PUBLICS:	Y or N: Y lists all global symbols with definitions.
LINE NUMBERS:	Y or N: Y includes line numbers in LIST file
STACK SIZE:	Non-zero decimal value sets stack size in RUN file
LOAD LOW:	Y or N: Y causes RUN file to be loaded in low memory when executed
DSALLOCATION:	Y or N: Y loads data at high-end of data segment

Notes:

1. For those prompts that require a Y or N, only the first character entered is interpreted and must be either Y, N or only the ENTER key. Any additional characters entered are ignored by the linker.

2. If you enter a *filespec* without specifying the drive, the default drive is assumed.

3. You may end (stop) the linker session prior to its normal end by pressing CTRL-BREAK.

Object Modules: Enter one or more *filespecs* for the object modules to be linked. If the extension is omitted, LINK assumes the filename extensions .OBJ or .REL. If an object module has any other filename extension, the extension must also be specified.

Filespecs must be separated by single commas (,) or by spaces.

LINK loads segments into classes in the order encountered.

If an object module is specified, but LINK cannot locate the file, a prompt requests the insertion of the diskette containing the specific module. This permits OBJ and REL files from several diskettes to be included.

To avoid a conflict with VM.TMP (which may have been allocated on the DOS default drive), these modules should be specified as residing on the other drive whose diskette can safely be exchanged. On a single-drive system, diskette exchanging can be done safely only if VM.TMP has not been opened. (A message will indicate if VM.TMP has been opened.)

RUN FILE: The *filespec* entered will be created to store the Run (executable) file that results from the LINK session. All Run files receive the filename extension .EXE, even if you specify an extension. (Your specified extension is ignored.)

As an option, you can respond to RUN FILE: with *filespec*/P. This tells LINK to display a message to the operator. This message requests the operator to insert the diskette that is to receive the Run file. If the /P option is used, the Run file should be explicitly directed to the non-default drive.

LIST FILE [run-filename.MAP]: The List file will be placed on the diskette that contains the Run file, unless overridden.

The List file contains an entry for each segment in the input (object) modules. Each entry also shows the offset (addressing) in the Run file.

If you do not enter a *filespec*, the Run filename with the default extension .MAP is used.

LIBRARIES [ ]: The valid responses are a list of library *filespecs* or simply the ENTER key. (Pressing only the ENTER key means no library search.) A library file is included as part of the compiler package.

When LINK attempts to reference a library file and cannot find it, a prompt requests that the drive identifier containing the library be entered.

From 1-8 library *filespecs* may be entered and must be separated by single commas or spaces.

LINK searches the library files in the order listed to resolve external references. When it finds the module that defines the external symbol, the module is processed as another object module.

PUBLICS [N]: The only valid responses begin with Y, N, or simply the ENTER key. (Pressing only the ENTER key defaults to N.)

A Y response directs LINK to list all public (global) symbols defined in the input modules. For each symbol, LINK lists its value and segment-offset location in the Run file. The symbols are listed at the end of the List file.

LINE NUMBERS [N]: The only valid responses begin with Y, N, or simply the ENTER key. (Pressing only the ENTER key defaults to N.)

A Y response directs LINK to include in the List file the line numbers and addresses of the source statements in the input modules.

STACK SIZE [0 bytes]: Valid responses are any positive decimal value up to 65536 bytes or simply the ENTER key.

If a value greater than 0 and less than 512 is specified, the value 512 is used.

This response is used to override the size of the stack that the assembler or compiler has provided for the load module being created. A response of 0 or just the ENTER key specifies that the original stack size provided by the assembler or compiler is to be used.

If the size of the stack is too small, the results of executing the resulting load module are unpredictable.

At least one input (object) module must contain a stack allocation statement. This is automatically provided by compilers. For the assembler, the source must contain a SEGMENT command that has the combine type of STACK. If a stack allocation statement was not provided, LINK returns a WARNING: NO STACK STATEMENT message.

LOAD LOW: [N]: The only valid responses begin with Y, N, or simply the ENTER key. (Pressing only the ENTER key defaults to N.)

A Y response to LINK will cause the loader to place the Run image as low as possible in memory. An N response to LINK will cause the loader to place the Run file as high as possible without overlaying the transient portion of COMMAND.COM, which occupies the highest area of memory when loaded.

Note: For PASCAL, the "Y" is required.

The LOAD LOW response is used by LINK in conjunction with the DSALLOCATION response (described below).

DSALLOCATION [N]: The only valid responses begin with Y, N, and simply the ENTER key. (Pressing only the ENTER key defaults to N.) If a group exists with a name of DGROUP, it will always be loaded lower than all other segments within the load module.

A Y response directs LINK to load all data, defined to be in DGROUP, at the **high-end** of the data segment. At run time, the data space starts at the lowest possible address that still allows addressability to all data elements of the group. If the LOAD LOW response is N (module loaded high), this allows any available memory below the

specifically allocated area within DGROUP to be allocated dynamically by your application and still be addressable by the same data space pointer.

> **Note:** The maximum amount of memory which can be dynamically allocated by the application will be 64K (or the amount actually available) minus the allocated portion of DGROUP.

An N response directs LINK to load all data, defined to be in the group whose group name is DGROUP, at the **low-end** of the data segment, beginning at an offset of 0. The only memory thus referenced by the data space pointer should be that specifically defined as residing in the group.

All other segments of any type in any GROUP other than DGROUP will be loaded at the low-end of their respective groups, as defined under the "N" response above.

Special Command Characters

LINK recognizes two special command characters.

& Use the ampersand (&) to extend the current line. To enter a large number of responses (each of which may be very long), enter an ampersand at the end of the line to extend the logical line, then continue to enter responses to the command prompt. Do not press the ENTER key until all responses for the current prompt are entered.

Use a single exclamation point (!) followed immediately by the ENTER key at any time after the first two prompts (from LIST FILE: on) to select default responses to the remaining prompts. This feature saves time and overrides the need to keep pressing the ENTER key.

> **Note:** Once the exclamation point has been entered, you can no longer respond to any of the remaining prompts for that linker session. Therefore, do not use the exclamation point to skip over some prompts. For this, use only the ENTER key.

How to Invoke the LINK Program

Linker Invocation

You can invoke the LINK program by using one of two options:

Option 1—Console Responses

From your keyboard, enter:

```
LINK
```

The linker is loaded into memory and displays a series of nine prompts, one at a time, to which you must enter the requested responses.

If you enter an erroneous response, such as the wrong *filespec* or an incorrectly spelled *filespec*, you must press CTRL-BREAK to exit LINK, then restart LINK. If the error has been typed but not entered, you may edit the erroneous characters, for that line only.

Option 2—Automatic Responses

From your keyboard, enter:

```
LINK filespec
```

For this option, you enter the *filespec* of an Automatic Response File.

Before using this option, you must create the Automatic Response File. It contains several lines of text, each of which is the response to a linker prompt. These responses must be in the same order as the linker prompts defined previously. If desired, a long response may be contained across several lines by using the ampersand to continue the same response onto the next line.

Use of the filename extension is optional and may be any name. (There is no default extension.)

Use of this option permits the LINK invocation command to be entered from the keyboard or within a batch file without requiring any operator response.

Angle brackets may be included in a response line to perform two functions:

1. The text contained within the brackets is treated as a remark which is displayed on the screen but otherwise ignored.

2. After the responses on the line containing the angle bracket comment have been acted upon by LINK, execution pauses and waits for a response from the keyboard. You may then enter additional items (ending them with the ENTER key), or you may just press the ENTER key. The linker continues processing with the next statement from the Automatic Response File.

For example, the Automatic Response File shown below contains:

- The names of standard modules (MODA. . .MODF)
- A prompt asking you for additional object module names <YOUR MODULE?>.
- Answers to the other eight prompts.

```
MODA,MODB,MODC&
MODD,MODE,MODF&
<YOUR MODULE?>
RUNFILE/P
PRINTOUT
PASCAL.LIB
Y
Y
O
N
Y
```

Notes:

1. In this example, the use of the ampersand causes the modules listed in the first two lines and any module entered by the operator in response to the <YOUR MODULE?> entry to be considered as the response to the OBJECT MODULES prompt.

2. Each of the above lines ends with the ENTER key.

Definitions

Segment, *Group*, and *Class* are terms that appear in some of the error messages in Appendix A. These terms describe the underlying function of LINK. An understanding of the concepts that define these terms provides a basic understanding of the way LINK works.

Segment

A *segment* is a contiguous area of memory up to 64K bytes in length. A segment may be located anywhere in memory on a "paragraph" (16-byte) boundary. Each of the four segment registers defines a segment. (The segments can overlap.) Each 16-bit address is an offset from the beginning of a segment. The contents of a segment are addressed by a segment register/offset pair.

The contents of various portions of the segment are determined at machine language generation time.

However, neither size nor location is necessarily fixed by the machine language generator because this portion of the segment may be combined at linker time with other portions forming a single segment.

A program's ultimate location in memory is determined at load time by the relocation loader facility provided in COMMAND. COM, based on your response to the LOAD LOW prompt.

The DEBUG Program

The DEBUG program can be used to:

- Provide a controlled testing environment so you can monitor and control the execution of a program to be debugged. You can fix problems in your program directly, and then execute immediately to determine that the problems are resolved. There is no need to reassemble a program to find out if your changes worked.
- Load, alter, or display any file.
- Execute **object files** (executable programs in machine language format).

This chapter explains how to use the DEBUG program. The information appears in the following order.

- How to invoke the DEBUG program
- Summary of the commands you use with DEBUG
- Definition of the DEBUG command parameters you supply
- Detailed description of the DEBUG commands

How to Invoke the DEBUG Program

To invoke DEBUG, enter:

DEBUG [*filespec*]

If you enter *filespec* the DEBUG program loads the specified file into memory. You may now enter commands to alter, display, or execute the contents of the specified file.

If you do **not** enter *filespec* you must either work with the present memory contents, or load the required file into memory. (Use the Name and Load commands.) Then you can enter commands to alter, display, or execute the memory contents.

When the DEBUG program starts, the registers and flags are set to the following values for the program being debugged.

- The segment registers (CS, DS, ES, and SS) are set to the bottom of free memory; that is, the first segment after the end of the DEBUG program.
- The Instruction Pointer (IP) is set to X'0100'.
- The Stack Pointer (SP) is set to the end of the segment or the bottom of the transient portion of COMMAND.COM, whichever is lower. The segment size at offset 6 is reduced by X'100' to allow for a stack of that size.
- The remaining registers (AX, BX, CX, DX, BP, SI, and DI) are set to zero. However, if you invoke the DEBUG program with a *filespec*, the CX register contains the length of the file in bytes.
- The flags are set to their cleared values. (See the Register command.)
- The default disk transfer address is set to X'80' in the code segment.

Notes:

1. If a file loaded by DEBUG has an extension of .EXE, DEBUG does the necessary relocation and sets the segment registers, stack pointer, and Instruction Pointer to the values defined in the file (with the exception that the DS and ES registers will point to the Program Segment Prefix at the lowest available segment). The CX register is set to zero.

   The program is loaded at the high end of memory if the appropriate parameter was specified when the linker created the file.

2. If a file loaded by DEBUG has an extension of .HEX, the file is assumed to contain ASCII representation of hexadecimal characters, and is converted to binary while being loaded.

Summary of DEBUG Commands

See "Format Notation" in Chapter 2 for an explanation of the notation used in the format of the following commands.

Command	Purpose	Format
Dump	Display memory	D [*address*] or D [*range*]
Enter	Change memory	E *address* [*list*]
Fill	Change memory blocks	F *range list*
Go	Execute with optional breakpoints	G [=*address*] [*address* [*address*. . .]]
Hexarithmetic	Hexadecimal add-subtract	H *value value*
Input	Read/display input byte	I *portaddress*
Load	Load file or absolute diskette sectors	L [*address* [*drive sector sector*]]
Move	Move memory block	M *range address*
Name	Define files and parameters	N *filespec* [*filespec*. . .]

Command	Purpose	Format
Output	Send output byte	O *portaddress byte*
Quit	End DEBUG program	Q
Register	Display registers/ flags	R [*registername*]
Search	Search for characters	S *range list*
Trace	Execute and display	T [=*address*] [*value*]
Unassemble	Unassemble instructions	U [*address*] or U [*range*]
Write	Write file or absolute diskette sectors	W [*address* [*drive sector sector*]]

The DEBUG Command Parameters

Parameter	Definition
address	Enter a 1- or 2-part designation in one of the following formats: • An alphabetic segment register designation, plus an offset value, such as: CS:0100 • A segment address, plus an offset value, such as: 4BA:0100 • An offset value only, such as: 0100 (In this case, each command uses a default segment.) **Notes:** 1. In the first two formats, the colon is required to separate the values. 2. All numeric values are hexadecimal and may be entered as 1-4 characters. 3. The memory locations specified in *address* must be valid; that is, they must actually exist. Unpredictable results will occur if an attempt is made to access a non-existent memory location.

Parameter	Definition
byte	Enter a 1 or 2 character hexadecimal value.
drive	Enter a single digit (for example, 0 for drive A or 1 for drive B) to indicate which drive data is to be loaded from or written to. (See the Load and Write commands.)
filespec	Enter a 1- to 3-part file specification consisting of a drive designation, a filename, and a filename extension. All three fields are optional. However, for the Name command to be meaningful, you must specify a drive designator, filename or both. (See the Name command.)
list	Enter one or more *byte* and/or *string* values. For example, E CS:100 F3 'XYZ' 8D 4 "abcd" has 5 items in the *list* (that is, three *byte* entries and two *string* entries having a total of 10 bytes).
portaddress	Enter a 1-4 character hexadecimal value to specify an 8- or 16-bit port address. (See the Input and Output commands.)

Parameter	Definition
range	Enter either of the following formats to specify the lower and upper addresses of a range.

- *address address*

  For example:

  CS:100 110

  Note: Only an offset value is allowed in the second address. The addresses must be separated by a space or comma.

- *address* L *value*

  where *value* is the number of bytes (in hexadecimal) to be processed by the command. For example:

  CS:100 L 11

  Notes:

  1. The limit for *range* is X'10000'. To specify that *value* within 4 hexadecimal characters, enter 0000 (or 0).

  2. The memory locations specified in *range* must be valid; that is, they must actually exist. Unpredictable results will occur if an attempt is made to access a non-existent memory location.

Parameter	Definition
sector sector	Enter 1-3 character hexadecimal values to specify: 1. The starting relative sector number and 2. The number of sectors to be loaded or written. In DEBUG, relative sectors are sequentially numbered from 0-13F, beginning at track 0 sector 1. The maximum number of sectors that can be loaded or written with a single command is X'80'. A sector contains 512 bytes. (See the Load and Write commands.)
registername	See the Register command.
string	Enter characters enclosed in quotation marks. The quotation marks can be either single (') or double ("). The ASCII values of the characters in the *string* are used as a *list* of byte values. Within a *string*, the **opposite** set of quotation marks can be used freely as characters. However, if the **same** set of quotation marks (as the delimiters) must be used within the *string*, then the quotation marks must be doubled. (The doubling does not appear in memory.)

Parameter	Definition
	For example: 1. 'This "literal" is correct' 2. 'This ' 'literal' ' is correct' 3. 'This 'literal' is not correct' 4. 'This " "literal" " is not correct' 5. "This 'literal' is correct" 6. "This " "literal" " is correct" 7. "This "literal" is not correct" 8. "This ' 'literal' ' is not correct" In the 2nd and 6th cases above the word *literal* is enclosed in one set of quotation marks in memory. In the 4th and 8th cases above, the word *literal* is not correct unless you really want it enclosed in two sets of quotation marks in memory.
value	Enter a 1-4 character hexadecimal value to specify that: • The numbers to be added and subtracted (see the Hexarithmetic command), or • The number of instructions to be executed by the Trace command, or • The number of bytes a command should operate on. (See the Dump, Fill, Move, Search, and Unassemble commands.)

The DEBUG Commands

This section presents a detailed description of how to use the commands to the DEBUG program. The commands appear in alphabetical order; each with its format, purpose, and example where appropriate.

The following general information applies to the DEBUG commands.

- A command is a single letter usually followed by one or more parameters.
- Commands and parameters can be entered in upper or lower case, or a mixture of both cases.
- Commands and parameters may be separated by delimiters (spaces or commas). Delimiters are only required, however, between two consecutive hexadecimal values. Thus, these commands are equivalent.

```
dcs:100 110
d cs:100 110
d,cs:100,110
```

- Commands can be ended by pressing CTRL-BREAK.
- Commands become effective only after the ENTER key is pressed.
- For commands producing a large amount of output, press CTRL-NUMLOCK to suspend the display in order to read it before it scrolls away. Press any other character to restart the display.

  The control-function and DOS editing keys, described in Chapter 1, apply while using the DEBUG program.

- If a syntax error is encountered, the line is displayed with the error pointed out as follows:

```
d cs:100 CS:110
           ^ error
```

  In this case, the Dump command is expecting the second address to contain only a hexadecimal offset value. It finds the S, which is not a valid hexadecimal character.

- The prompt from the DEBUG program is a hyphen (-).

Dump Command

Format: D [*address*]

or

D [*range*]

Purpose: Display the contents of a portion of memory.

The dump is displayed in two parts:

1. A hexadecimal portion (Each byte is displayed in hexadecimal), and

2. An ASCII portion. (The bytes are displayed as ASCII characters.) Unprintable characters are indicated by a period (.).

With a 40-column system display format, each line begins on an 8-byte boundary and shows 8 bytes.

With an 80-column system display format, each line begins on a 16-byte boundary and shows 16 bytes. There is a hyphen between the 8th and 9th bytes.

> **Note:** The first line may have fewer than 8 or 16 bytes if the starting address of the dump is not on a boundary. In this case, the second line of the dump begins on a boundary.

Option 1

D

or

D *address*

Dump Command

Display the contents of X'40' bytes (40-column mode) or X'80' bytes (80-column mode).

The contents are dumped starting with the specified *address.*

If no address is given, the D command assumes the starting address is the location following the last location displayed by a previous D command. Thus, it is possible to dump consecutive 40-byte or 80-byte areas by entering consecutive D commands without parameters.

If no previous D command was entered, the location is offset X'0100' into the segment originally initialized in the segment registers by DEBUG.

> **Note:** If you enter only an offset for the starting address, the D command assumes the segment contained in the DS register.

Option 2

D *range*

Display the contents of the specified address range.

> **Note:** If you enter only an offset for the starting address, the D command assumes the segment contained in the DS register. If you specify an ending address, enter it with only an offset value.

For example:

```
D cs:100 10B
```

A 40-column display format might look like this:

```
04BA:0100  42 45 52 54 41 20 54 00
                      BERTA T.

04BA:0108  20 42 4F 52
                        BOR
```

Enter Command

Format: E *address* [*list*]

Purpose: The Enter command has two modes of operation.

- Replace the contents of one or more bytes, starting at the specified address, with the values contained in the *list*. (See Option 1.)
- Display and allow modification of bytes in a sequential manner. (See Option 2.)

> **Note:** If you enter only an offset for the address, the E command assumes the segment contained in the DS register.

Option 1

E *address list*

The *list* is placed in memory beginning at the specified *address*.

For example:

```
E ds:100 F3 "xyz" 8D
```

Memory locations ds:100 through ds:104 are filled with the five bytes specified in the *list*.

Enter Command

Option 2

E *address*

The address and the byte at that location are displayed, then the system waits for your input. You now can take any of four actions.

1. Enter a 1 or 2 character hexadecimal value to replace the contents of the byte; then take any of the next three actions.

2. Press the space bar to advance to the next address. Its contents are displayed. If you want to change those contents take action 1, above.

   To advance to the next byte without changing the current byte, press the space bar again.

3. Enter a hyphen (-) to back up to the preceding address. A new line is displayed with the preceding address and its contents. If you want to change those contents take action 1, above.

   To back up one more byte without changing the current byte, enter another hyphen.

4. Press the ENTER key to end the Enter command.

   Note: Display lines can have 4 or 8 bytes of data, depending on whether the system display format is 40- or 80-column. Spacing beyond an 8-byte boundary causes a new display line, with the beginning address, to be started.

For example:

```
E cs:100
```

might cause this display:

```
04BA:0100 EB._
```

To change the contents of 04BA:0100 from X'EB' to X'41', enter 41.

```
04BA:0100 EB.41_
```

To see the contents of the next three locations, press the space bar three times. The screen might look like this:

```
04BA:0100  EB.41  10.  00.  BC._
```

To change the contents of the current location (04BA:0103) from X'BC' to X'42', enter 42.

```
04BA:0100  EB.41  10.  00.  BC.42_
```

Now, suppose you want to back up and change the X'10' to X'6F'. This is what the screen would look like after entering two hyphens and the replacement byte.

```
04BA:0100  EB.41  10.  00.  BC.42-
04BA:0102  00.-
04BA:0101  10.6F
```

Press the ENTER key to end the Enter command. You will see the hyphen (-) prompt.

Fill Command

Format: F *range list*

Purpose: Fill the memory locations in the *range* with the values in the *list*.

If the *list* contains fewer bytes than the address range, the list is used repeatedly until all the designated memory locations are filled.

If the *list* contains more bytes than the address range, the extra *list* items are ignored.

> **Note:** If you enter only an offset for the starting address of the *range*, the Fill command assumes the segment contained in the DS register.

Example:

```
F 4BA:100 L 5 F3 "XYZ" 8D
```

Memory locations 04BA:100 through 04BA:104 are filled with the 5 bytes specified. Remember that the ASCII values of the *list* characters are stored. Thus, locations 100-104 will contain F3 58 59 5A 8D.

Go Command

Format: G [=*address*] [*address* [*address*. . .]]

Purpose: This command:

1. Executes the program you are debugging, and optionally:

2. Stops the execution when the instruction at a specified *address* is reached (breakpoint), and displays the registers, flags, and the next instruction to be executed.

Program execution begins with the current instruction, whose address is determined by the contents of the CS and IP registers, unless overridden by the =*address* parameter (the = must be entered). If =*address* is specified, program execution begins with CS:=*address*.

There are two options with the Go command.

Option 1

G [=*address*]

Entering the G command without breakpoints simply executes the program you are debugging. This is useful when testing program execution with different parameters each time. (See the Name command.) Be certain the CS:IP values are set properly before issuing the G command, if not using =*address*.

Go Command

Option 2

G [=*address*] *address*
[*address*. . .]

This option performs the same function as Option 1 but, in addition, allows breakpoints to be set at the specified addresses. (This is a method of causing execution to stop at a specified location so the system/program environment can be examined.)

Up to ten breakpoints can be specified in any order. (You may wish to take advantage of this if your program has many paths, and you want to stop the execution no matter which path the program takes.)

The DEBUG program replaces the instruction codes at the breakpoint addresses with an interrupt code (X'CC'). If **any one** breakpoint is reached during execution, the execution is stopped, the registers and flags are displayed, and all the breakpoint addresses are restored to their original instruction codes. If no breakpoint is reached, the instructions are **not** restored.

Notes:

1. Be certain that the *address* parameters refer to locations that contain valid 8086 instruction codes. If you specify an *address* that does not contain the first byte of a valid instruction, unpredictable results will occur.

2. The stack pointer must be valid and have six bytes available for the Go command, otherwise, unpredictable results will occur.

3. If only an offset is entered for a breakpoint, the G command assumes the segment contained in the CS register.

For example:

```
G 102 1EF 208
```

Execution begins with the current instruction, whose address is the current values of CS:IP. (The =*address* parameter was not used.)

Three breakpoints are specified. Assume that the second is reached. Execution stops before the instruction at location CS:1EF is executed; the original instruction codes are restored; the display occurs, and the Go command ends.

See the Register command for a description of the display.

Hexarithmetic Command

Format: H *value value*

Purpose: Add the two hexadecimal values, then subtract the second from the first.

Display the sum and difference on one line.

Example:

```
H 0F 8
17 07
```

The hexadecimal sum of 000F and 0008 is 0017, and their difference is 0007.

Input Command

Format: I *portaddress*

Purpose: Input and display (in hexadecimal) one byte from the specified port.

Example:

```
I 2F8
6B
```

The single hexadecimal byte read from port 02F8 is displayed (6B).

Load Command

Format: L [*address* [*drive sector sector*]]

Purpose: Load a file or absolute diskette sectors into memory.

The maximum number of sectors that can be loaded with a single Load command is X'80'.

> **Note:** DEBUG displays a message if a disk read error occurs. The read operation can be retried by pressing F3 to re-display the L command, then the ENTER key.

There are two options with the Load command.

Option 1

L *address drive sector sector*

Data is loaded from the diskette specified by *drive*, and placed in memory beginning at the specified *address*. The data is read from the specified starting relative sector (first *sector*) and continues until the requested number of sectors is read (second *sector*).

> **Note:** If only an offset is entered for the beginning address, the L command assumes the segment contained in the CS register.

For example, to load data, you might enter:

```
L 4BA:100 1 0F 6D
```

The data is loaded from the diskette in drive B and placed in memory beginning at 4BA:100. 109 consecutive sectors of data are transferred, starting with relative sector 15 (the 16th sector on the diskette).

Option 2

L

or

L *address*

When issued without parameters (or with only the *address* parameter), the Load command loads the file whose filespec is properly formatted in the file control block at CS:5C.

This condition is met by specifying the filespec when invoking the DEBUG program, or by using the Name command. (Note that if DEBUG was invoked with a filespec and subsequent Name commands were used, a new Name command may have to be entered for the proper filespec before issuing the Load command.)

The file is loaded into memory beginning at CS:100 (or the location specified by *address*) and is read from the drive specified in the filespec (or from the default drive, if none was specified).

The CX register is set to the number of bytes read (except for files with an extension of .EXE).

> **Note:** To load files with extensions of .EXE and .HEX, see the notes in "How to Invoke the DEBUG Program" at the beginning of this chapter.

For example:

```
DEBUG
-N myprog
-L
-
```

The file named 'myprog' is loaded from the default diskette and placed in memory beginning at location CS:0100.

Move Command

Format: M *range address*

Purpose: Move the contents of the memory locations specified by *range* to the locations beginning at the *address* specified.

Overlapping moves (The source and destination areas share some of the same memory locations.) are always performed without loss of data during the transfer.

The data in the source area remains unchanged unless overwritten by the move.

Notes:

1. If only an offset is entered for the beginning address of the range, the M command assumes the segment contained in the DS register. If you specify an ending address for the range, enter it with only an offset value.

2. If only an offset is entered for the address of the destination area, the M command assumes the segment contained in the DS register.

Example:

```
M CS:100 110 500
```

The 11 bytes of data from CS:100 through CS:110 are moved to the area of memory beginning at DS:500.

Name Command

Format: N *filespec* [*filespec. . .*]

Purpose: The Name command has two functions:

1. Format file control blocks for the first two filespecs, at CS:5C and CS:6C. (Invoking DEBUG with a *filespec* also formats a file control block at CS:5C.)

   The file control blocks are set up for the use of the Load and Write commands, and to supply required filenames for the program being debugged.

2. All specified filespecs and other parameters are placed (exactly as entered, including delimiters) in a parameter save area at CS:81, with CS:80 containing the number of characters entered.

   Note: If the DEBUG program is started without a *filespec*, the Name command must be used before a file can be loaded with the L command.

Name
Command

Example:

```
DEBUG
-N myprog
-L
-
```

To define filespecs or other parameters required by the program being debugged, enter:

```
DEBUG myprog
-N file1 file2
-
```

In this example, DEBUG loads the file **myprog** at CS:100, and leaves the file control block at CS:5C formatted with the same filespec. Then, the Name command formats file control blocks for **file1** and **file2** at CS:5C and CS:6C, respectively. (The file control block for **myprog** is overwritten.)

The parameter area at CS:81 contains all characters entered after the "N", including all delimiters, and CS:80 contains the count of those characters (X'0C').

Output Command

Format: O *portaddress byte*

Purpose: Send the *byte* to the specified output port.

Example: `O 2F8 4F`

The byte value 4F is sent to output port 2F8.

Quit Command

Format: Q

Purpose: End the DEBUG program.

The file in memory that you are working on is **not** saved by the action of the Quit command. You must have used the Write command for this purpose.

DEBUG returns to the command processor which then issues the normal command prompt.

Example:

```
-Q
A>
```

Register Command

Format: R [*registername*]

Purpose: The Register command has three functions:

- Displays the hexadecimal contents of a single register, with the option of changing those contents, or
- Displays the hexadecimal contents of all the registers, plus the alphabetic flag settings, and the next instruction to be executed, or
- Displays the eight 2-letter alphabetic flag settings, with the option of changing any or all of them.

> **Note:** When the DEBUG program starts, the registers and flags are set to certain values for the program being debugged. (See "How to Invoke the DEBUG Program" at the beginning of this chapter.)

Display a Single Register

The valid **registernames** are:

AX	BP	SS
BX	SI	CS
CX	DI	IP
DX	DS	PC
SP	ES	F

(Both IP and PC refer to the instruction pointer.)

For example, to display the contents of a single register, you might enter:

```
R AX
```

and the system might respond with

```
AX F1E4
:_
```

Now you may take one of two actions: press ENTER to leave the contents unchanged, or change the contents of the AX register by entering a 1-4 character hexadecimal value, such as X'FFF'

```
AX F1E4
:FFF_
```

Now pressing ENTER changes the contents of the AX register to X'0FFF'.

Display All Registers and Flags

To display the contents of all registers and flags (and the next instruction to be executed), enter:

```
R
```

and the system might respond with:

```
AX=0E00 BX=00FF CX=0007 DX=01FF
SP=039D BP=0000 SI=005C DI=0000
DS=04BA ES=04BA SS=04BA CS=04BA
IP=011A  NV UP DI NG NZ AC PE NC
04BA:011A  CD21           INT     21
```

The first four lines display the hexadecimal contents of the registers and the eight alphabetic flag settings. The last line indicates the location of the next instruction to be executed, and its hexadecimal and unassembled formats. This is the instruction pointed to by CS:IP.

Note: A system with an 80-column display shows:

1st line – 8 registers
2nd line – 5 registers and 8 flag settings
3rd line – next instruction information

A system with a 40-column display shows:

1st line – 4 registers
2nd line – 4 registers
3rd line – 4 registers
4th line – 1 register and 8 flag settings
5th line – next instruction information

Display All Flags

There are eight flags, each with 2-letter codes to indicate either a "set" condition or a "clear" condition.

The flags appear in displays in the same order as presented in the following table.

Flag name		Set	Clear
Overflow	(yes/no)	OV	NV
Direction	(decrement/increment)	DN	UP
Interrupt	(enable/disable)	EI	DI
Sign	(negative/positive)	NG	PL
Zero	(yes/no)	ZR	NZ
Auxiliary carry	(yes/no)	AC	NA
Parity	(even/odd)	PE	PO
Carry	(yes/no)	CY	NC

Register Command

To display all flags, enter:

```
R F
```

If all the flags were in a "set" condition, the response would be:

```
OV DN EI NG ZR AC PE CY - _
```

Now you may take one of two actions: press ENTER to leave the settings unchanged, or change any or all of them.

To change a flag, just enter its opposite code. The opposite codes can be entered in any order and with or without intervening spaces. For example, to change the 1st, 3rd, 5th, and 7th flags enter:

```
OV DN EI NG ZR AC PE CY - PONZDINV
```

(They are entered in reverse order in this example.)

Press ENTER and the flags are modified as specified, the prompt appears, and you can enter the next command

If you want to see that the new codes are in effect, enter:

```
R F
```

and the response will be:

```
NV DN DI NG NZ AC PO CY - _
```

The 1st, 3rd, 5th, and 7th flags are changed as requested The 2nd, 4th, 6th, and 8th flags are unchanged.

> **Note**: A single flag can be changed only once per R F command.

Search Command

Format: S *range list*

Purpose: Search the *range* for the character(s) in the *list*.

All matches are indicated by displaying the addresses where matches are found.

A display of the prompt (-) without any addresses means that no match was found.

> **Note:** If you enter only an offset for the starting address of the range, the S command assumes the segment contained in the DS register.

Example:

```
S CS:100 110 41
```

The range of addresses from CS:100 through CS:110 is searched for X'41'. If two matches are found the response might be:

```
04BA:0104
04BA:010D
```

As another example:

```
S CS:100 L 11 41 "AB" E
```

The same range of addresses as in the previous example is searched for a match with the 4-byte-long list. The starting addresses of all matches are listed. If no match is found, no address is displayed.

Trace Command

Format: T [=*address*] [*value*]

Purpose: Execute one or more instructions starting with the instruction at CS:IP, or at =*address* if it is specified. (The = must be entered.) One instruction is assumed, but you can specify more than one with *value*.

Display the contents of all registers and flags **after each** instruction executes. (For a description of the display format, see the Register command.)

> **Note:** The display caused by the Trace command continues until *value* instructions are executed. Therefore, when tracing multiple instructions, remember that you can suspend the scrolling at any time by pressing CTRL-NUMLOCK. Resume scrolling by entering any other character.

Example:

```
T
```

If the IP register contains 011A, and that location contains B40E (MOV AH,0EH), this might be returned:

```
AX=0E00 BX=00FF CX=0007 DX=01FF
SP=039D BP=0000 SI=005C DI=C000
DS=04BA ES=04BA SS=04BA CS=04BA
IP=011C  NV UP DI NG NZ AC PE NC
04BA:011C  CD21           INT     21
```

This displays the results **after** the instruction at 011A is executed, and indicates the next instruction to be executed is the INT 21 at location 04BA:011C.

```
T 10
```

Trace Command

Sixteen instructions are executed (starting at CS:IP). The contents of all registers and flags are displayed after each instruction. The display stops after the 16th instruction has been executed. Displays may scroll off the screen unless suspended by the CTRL-NUMLOCK keys.

Unassemble Command

Format: U [*address*]

or

U [*range*]

Unassemble instructions and display their addresses and hexadecimal values, together with assembler-like statements. For example, a display might look like:

```
04BA:0100  206472    AND [SI+72],AH
04BA:0103  FC        CLD
04BA:0104  7665      JBE 016B
```

The number of bytes unassembled depends on your system display format (whether 40 or 80 columns), and which option you use with the Unassemble command.

Notes:

1. In all cases, the number of bytes unassembled and displayed may be slightly more than either the amount requested or the default amount. This happens because the instructions are of variable lengths; therefore, to unassemble the last instruction may include more bytes than expected.

2. Be certain that the *address* parameters refer to locations that contain valid 8086 instruction codes. If you specify an *address* that does not contain the first byte of a valid instruction, the display will be erroneous.

3. If you enter only an offset for the starting address, the U command assumes the segment contained in the CS register.

There are two options with the Unassemble command.

Unassemble Command

Option 1

U

or

U *address*

16 bytes are unassembled with a 40-column display. 32 bytes are unassembled with an 80-column display.

Instructions are unassembled beginning with the specified *address.*

If no *address* is specified, the U command assumes the starting address is the location following the last instruction unassembled by a previous U command. Thus, it is possible to unassemble consecutive locations, producing continuous unassembled displays, by entering consecutive U commands without parameters.

If no previous U command was entered, the location is offset X'0100' into the segment originally initialized in the segment registers by DEBUG.

Option 2

U *range*

All instructions in the specified address range are unassembled, regardless of the system display format.

> **Note:** If you specify an ending address, enter it with only an offset value.

Unassemble Command

For example:

```
U 04ba:0100 108
```

The display response might be:

```
04BA:0100  206472      AND [SI+72],AH
04BA:0103  FC          CLD
04BA:0104  7665        JBE 016B
04BA:0106  207370      AND [BP+DI+70],DH
```

The same display would appear if you entered:

```
U 04BA:100 L 7
```

or

```
U 04BA:100 L 8
```

or

```
U 04BA:100 L 9
```

Write Command

Format: W [*address* [*drive sector sector*]]

Purpose: Write the data being debugged to diskette.

The maximum number of sectors that can be written with a single Write command is X'80'.

> **Note:** DEBUG displays a message if a disk write error occurs. The write operation can be retried by pressing F3 to re-display the Write command, then the ENTER key.

There are two options with the Write command.

Option 1

W *address drive sector sector*

The data beginning at the specified *address* is written to the diskette in the indicated *drive*. The data is written starting at the specified starting relative sector (first *sector*) and continues until the requested number of sectors are filled (second *sector*).

> **Notes:**
>
> 1. Be extremely careful when writing data to absolute sectors because an erroneous sector specification will destroy whatever was on the diskette at that location.
>
> 2. If only an offset is entered for the beginning address, the W command assumes the segment contained in the CS register.

For example:

```
W 1FD 1 100 9
```

The data beginning at CS:01FD is written to the diskette in drive B, starting at relative sector 100 and continuing for 9 sectors.

Option 2

W

or

W *address*

When issued without parameters (or with only the *address* parameter), the W command writes the file (whose filespec is properly formatted in the file control block at CS:5C) to diskette.

This condition is met by specifying the filespec when invoking the DEBUG program, or by using the Name command. (Note that if DEBUG was invoked with a filespec and subsequent Name commands were used, a new Name command may have to be entered for the proper filespec before issuing the Write command.)

In addition, the CX register must be set to the number of bytes to be written. It may have been set properly by the DEBUG or Load commands, but might have been changed by a Go or Trace command. You must be certain the CX register contains the correct value.

The file beginning at CS:100 (or the location specified by *address*) is written to the diskette in the drive specified in *filespec* or the default drive if none was specified.

Write Command

The debugged file is written over the original file that was loaded into memory, or into a new file if the filename in the FCB didn't previously exist.

> **Note:** An error message is issued if you try to write a file with an extension of .EXE or .HEX. These files must be written in a specific format that DEBUG cannot support.
>
> If it is necessary to modify a file with an extension of .EXE or .HEX, and the exact locations to be modified are known, it can be done as follows:
>
> 1. RENAME the file to an extension other than .EXE or .HEX.
>
> 2. Load the file into memory using the DEBUG or Load command.
>
> 3. Modify the file as needed in memory, but do not try to execute it with the Go or Trace commands. Unpredictable results would occur.
>
> 4. Write the file back using the Write command.
>
> 5. RENAME the file back to its correct name.

Appendix A. Messages

The first word of each message description is the name of the program or command that generated the message.

Allocation error for file *filename*

CHKDSK. An invalid sector number was found in the file allocation table. The file was truncated at the end of the last valid sector.

Attempt to access data outside of segment bounds

LINK. An object file is probably invalid.

Attempted write-protect violation

FORMAT. The diskette being formatted cannot be written on because it is write-protected. You are prompted to insert a new diskette and press a key to restart formatting.

Aux I/O error

DOS. An input or output error occurred while trying to use the first Asynchronous Communications Adapter.

Bad command or file name

DOS. The command just entered is not a valid internal command, and a file called *command-name*.COM or *command-name*.EXE could not be found on the specified (or default) drive.

Bad or missing Command Interpreter

STARTUP. The diskette in drive A does not contain a copy of COMMAND.COM, or an error occurred while loading it. If System Reset fails to solve the problem, copy COMMAND.COM from a backup diskette to the diskette that failed.

BF

DEBUG. Bad flag. An invalid flag code setting was specified. Try the Register (R F) command again with the correct code.

BP

DEBUG. Breakpoints. More than ten breakpoints were specified for the Go command. Try the Go (G) command again with ten or fewer breakpoints.

BR

DEBUG. Bad register. An invalid register name was specified. Try the Register (R) command again with a correct register name.

XXXXXXXXXX bytes of disk space freed

CHKDSK. Diskette space marked as allocated was found not to be allocated. Therefore, the space was freed and made available.

Cannot compare file to itself

COMP. The two filenames entered refer to the same file on the same diskette. COMP assumes an error was made in entering one of the filenames.

Cannot edit .BAK file—rename file

EDLIN. .BAK files are considered to be backup files, with more up-to-date versions of the files assumed to exist. Therefore, .BAK files normally shouldn't be edited.

If it is necessary to edit the .BAK file, either rename the file, or copy it and give the copy a different name.

Cannot open temporary file

LINK. The directory is full.

Compare error at offset XXXXXXXX

COMP. The files being compared contain different values at the displayed offset (in hexadecimal) into the file. The differing values are also displayed in hexadecimal.

Compare error(s) on track nn

DISKCOMP. One or more locations on the indicated track contain differing information between the diskettes being compared.

Data error reading drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x".

Data error writing drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x".

DF

DEBUG. Double flag. Conflicting codes were specified for a single flag. A flag can be changed only once per Register (R F) command.

Directory error-file: *filename*

CHKDSK. No valid sectors were allocated to the file. The *filename* is removed from the directory.

Disk boot failure

DOS. An error occurred while trying to load DOS into memory. If subsequent attempts to start the system also fail, use a backup DOS diskette.

Disk error reading drive x
Abort, Retry, Ignore?

DOS. A disk read or disk write error has occurred. The operation was retried three times without success. The system now waits until **one** of the following responses is made.

- Enter A for Abort. The system ends the program that requested the disk read or write.
- Enter R for Retry. The system tries the disk read or write operation again.
- Enter I for Ignore. The system pretends the error did not occur and continues the program.

To recover from an error condition, the responses are generally made in the following order:

R to retry the operation because the error may not reoccur.

I to ignore the error condition and continue the program.

A to abort the program.

> **Note:** When executing DEBUG, the second line of the message does not appear. To retry the disk operation, press F3 to re-display the Load or Write command, and then press the ENTER key.

Disk error writing drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x".

Disk full—file write not completed

EDLIN. An End Edit command was aborted because the diskette does not have enough free space to save the entire file.

Some of the file may be saved on diskette but the portion in memory not saved is lost.

Disk unsuitable for system disk

FORMAT. A defective track was detected where the DOS files were to reside. The diskette can be used only for data.

Diskette not initialized

CHKDSK. During its analysis of the diskette, CHKDSK could not recognize the directory or file allocation table. The diskette should be formatted again before further use. (It may be possible first to copy files to another diskette in order to preserve as much data as possible.)

Divide overflow

DOS. A program attempted to divide a number by zero, or the program had a logic error that caused an internal malfunction. The system simulates CTRL-BREAK processing.

Dup record too complex

LINK. A problem exists in an object module created from an assembler source program. Correct the machine language source program and then rerun LINK.

Duplicate file name or
File not found

RENAME. You tried to rename a file to a filename that already exists on the diskette, or the file to be renamed could not be found on the specified (or default) drive.

End batch job (Y/N)?

DOS. This message appears when you press CTRL-BREAK while DOS is processing a batch file. Press Y to stop processing the batch file. Pressing N only ends the command that was executing when CTRL-BREAK was pressed; processing resumes with the next command in the batch file.

Enter primary file name
Or strike the ENTER key to end

COMP. Enter the *filespec* of the first of two files to be compared.

Enter 2nd file name or drive id

COMP. Enter the *filespec* of the second of two files to be compared, or just the drive designator if the filename is the same as the primary filename.

Entry error

EDLIN. Correct the syntax error on the last command.

Eof mark not found

COMP. An unsuccessful attempt was made to locate the end of valid data in the last block of the files being compared. This message normally occurs when comparing non-text files, but should not occur when comparing text files.

Error in EXE file

DOS. An error was detected in the relocation information placed in the file by the LINK program.

Error in EXE/HEX file

DEBUG. The file contained invalid records or characters.

EXE/HEX file cannot be written

DEBUG. The data would require a backwards conversion that DEBUG doesn't support.

File allocation table bad, drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x". If this error persists, the disk is unusable and should be formatted again.

File cannot be copied onto itself

DOS. A request is made to COPY a file and place the copy (with the same name) on the same diskette as the original. Either change the name given to the copy or put it on another diskette.

File creation error

DOS. An unsuccessful attempt was made to add a new filename to the directory. Run CHKDSK to determine if the directory is full, or if some other condition caused the error.

File n empty

COMP. File n can represent either the first or second filename entered. In either case, the file contains no valid data.

File n not found

COMP. The first or second file specified does not exist on the specified (or default) drive.

File not found

DEBUG and DOS. A file named in a command or command parameter does not exist on the diskette in the specified (or default) drive.

File size error for file *filename*

CHKDSK. The file size shown in the directory is different from the actual size allocated for the file. The size in the directory is adjusted, up or down, to show the actual size (rounded to a 512-byte boundary).

Files are different sizes

COMP. The sizes of the files to be compared do not match. The compare cannot be done because one of the files contains data which the other does not.

Files cross-linked:
filename* and *filename

CHKDSK. The same data block is allocated to both files. No corrective action is taken automatically; so, you must correct the problem. For example:

1. Make copies of both files (use COPY command).
2. Delete the original files (use ERASE command).
3. Review the files for validity and edit as necessary.

Fixup offset exceeds field width

LINK. A machine language processor instruction refers to an address with a NEAR attribute instead of a FAR attribute. Edit assembler source program and process again.

Format failure

FORMAT. A disk error was encountered during the formatting process. The diskette is unusable.

Illegal Device Name

MODE. The specified printer must be LPT1:, LPT2:, or LPT3:

Incompatible system size

SYS. The target diskette contained a copy of DOS that is smaller than the one being copied. The system transfer does not take place. A possible solution might be to format a blank diskette (use the FORMAT /S command) and then copy any files to the new diskette.

Insert disk with batch file
and strike any key when ready

DOS. The diskette that contained the batch file being processed was removed. The batch processor is trying to find the next command in the file. Processing will continue when you insert the diskette in the appropriate drive and press a key.

Insert DOS disk in drive n
and strike any key when ready

DOS and FORMAT. Either DOS is trying to reload the command processor, or FORMAT is trying to load the DOS files, but the default drive does not contain the DOS diskette.

Insufficient disk space

DOS. The diskette does not contain enough free space to contain the file being written. If you suspect this condition is invalid, run CHKDSK to determine the status of the diskette.

Insufficient memory

DEBUG, DISKCOMP, DISKCOPY, and EDLIN. The amount of available memory is too small to allow these commands to function.

Insufficient space on disk

DEBUG. A Write command was issued to a diskette that doesn't have enough free space to hold the data being written.

Invalid COMMAND.COM in drive n

DOS. While trying to reload the command processor, the copy of COMMAND.COM on the diskette was found to be an incorrect version. You are prompted to insert a correct DOS diskette and press a key to continue.

Invalid date

DATE. An invalid date or delimiter was entered. The only valid delimiters in a date entry are the hyphen (-) and slash (/).

Invalid drive specification

DOS and commands. An invalid drive specification was just entered in a command or one of its parameters.

Invalid numeric parameter

LINK. Numeric value not in digits.

Invalid object module

LINK. Object module(s) incorrectly formed or incomplete (as when the language processor was stopped in the middle).

Invalid parameter

CHKDSK, DISKCOMP, DISKCOPY, FORMAT, and SYS. The parameter entered for these commands was not a drive specifier. Be sure to enter a valid drive specifier followed by a **colon**.

Invalid time

TIME. An invalid time or delimiter was entered. The only valid delimiters are the colon between the hours and minutes, and the minutes and seconds, and a period between the seconds and hundredths.

Invalid Y/N parameter

LINK. Response to a prompt did not begin with Y, N, or simply the ENTER key.

Line too long

EDLIN. Upon replacing a *string*, the replacement causes the line to expand beyond the 253-character limit. The Replace Text command is aborted.

Split the long line into shorter lines and then retry the Replace Text command.

Missing file name

RENAME. The second of the two required filenames is not specified.

No room for system on destination disk

SYS. The destination diskette did not already contain the required reserved space for DOS; therefore, the system cannot be transferred. A possible solution would be to format a blank diskette (using the FORMAT /S command), and then copy any other files to the new diskette.

No room in directory for file

EDLIN. The specified diskette already contains the maximum of 64 files.

No room in disk directory

DEBUG. The diskette in the drive specified by the Write command already contains the maximum of 64 files.

Non-System disk or disk error
Replace and strike any key when ready

STARTUP. There is no entry for IBMBIO.COM or IBMDOS.COM in the directory, or a disk read error occurred while starting up the system. Insert a DOS diskette in the drive.

Not found

EDLIN. Either the specified range of lines does not contain the *string* being searched for by the Replace Text or Search Text commands, or if a search is resumed by replying N to the "OK?" prompt, no further occurrences of the *string* were found.

Not ready error reading drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x". In this case, no retries are performed.

Not ready error writing drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x" In this case, no retries are performed.

Out of paper

DOS. Either the printer is out of paper or not powered ON.

Out of space on list file

LINK. This error is usually caused when there is not enough disk space for the List file.

Out of space on run file

LINK. This error is usually caused when there is not enough disk space for the Run file (.EXE).

Out of space on VM.TMP

LINK. No more disk space remained to expand the VM.TMP file.

Printer error

MODE. The MODE command (option 1) was unable to set the printer mode because of an I/O error, out of paper (or POWER OFF), or time out (not ready) condition.

Printer fault

DOS. The printer cannot accept data because it is offline.

Program size exceeds capacity of LINK

LINK. Load module is too large for processing.

Program too big to fit in memory

DOS. The file containing the external command cannot be loaded because it is larger than the available memory.

Requested stack size exceeds 64K

LINK. Specify a size $\leq$ 64K bytes when the STACK SIZE: prompt appears.

Sector not found error reading drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x".

Sector not found error writing drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x".

Seek error reading drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x".

Seek error writing drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x".

Segment size exceeds 64K

LINK. Attempted to combine identically named segments which resulted in segment requirement of greater than 64K bytes. 64K bytes is the addressing limit.

Symbol defined more than once

LINK. The Linker found two or more modules that define a single symbol name.

Symbol table capacity exceeded

LINK. Very many, very long names were entered that exceeded approximately 50K bytes. Use shorter and/or fewer names.

'Target' diskette unusable

DISKCOPY. This message follows an unrecoverable read, write, or verify error message. The copy on the 'target' diskette is incomplete because of the unrecoverable I/O error.

'Target' diskette write protected
Correct, then strike any key

DISKCOPY. You are trying to produce a copy on a diskette that is write-protected.

Too many external symbols in one module

LINK. The limit is 256 external symbols per module.

Too many groups

LINK. The limit is 256 groups.

Too many libraries specified

LINK. The limit is 8 libraries.

Too many public symbols

LINK. The limit is 1024 public symbols.

Too many segments or classes

LINK. The limit is 256 (segments and classes taken together).

Track 0 bad-disk unusable

FORMAT. Track 0 is where the boot record, file allocation table, and directory must reside.

Unexpected end of file on VM.TMP

LINK. The diskette containing VM.TMP has been removed.

Unrecoverable read error on drive x

DISKCOMP. Six attempts were made to read the data from the diskette in the specified drive.

Unrecoverable read error on 'source' disk

DISKCOPY. Six attempts were made to read the data from the 'source' diskette. DISKCOPY is unable to continue; so, the copy is incomplete.

Unrecoverable verify error on 'target' disk

DISKCOPY. Six attempts were made to verify the write operation to the 'target' diskette. DISKCOPY is unable to continue; so, the copy is incomplete.

Unrecoverable write error on 'target' disk

DISKCOPY. Six attempts were made to write the data to the 'target' diskette. DISKCOPY is unable to continue; so, the copy is incomplete.

Unresolved externals: *list*

LINK. The external symbols listed were not defined in the modules or library files that you specified.

Warning: no stack segment

LINK. None of the object modules specified contains a statement allocating stack space, but you responded with a non-zero entry to the STACK SPACE: prompt.

Write error

FORMAT. A write error occurred while writing the boot record or system files.

Write fault error writing drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x".

Write protect error writing drive x
Abort, Retry, Ignore?

DOS. See the message "Disk error reading drive x".

10 Mismatches–aborting compare

COMP. Ten mismatched locations were detected in the files being compared. COMP assumes the files are so different that further comparisons would serve no purpose.

Appendix B. DOS Technical Information

Appendixes B-E are intended to supply technically oriented users with information about the structure, facilities, and program interfaces of DOS. It is assumed that the reader is familiar with the 8086 architecture, interrupt mechanism, and instruction set.

DOS Structure

DOS consists of the following three components.

1. The boot record resides on the first sector of every diskette formatted by the FORMAT command. It is put on all diskettes in order to produce an error message if you try to start up the system with a non-DOS diskette in drive A.

2. The Read-Only Memory (ROM) BIOS interface module (file IBMBIO.COM) provides a low-level interface to the ROM BIOS device routines. It also contains routines to trap and report, via console messages, divide-by-zero, printer out-of-paper, and Asynchronous Communications Adapter error situations.

3. The DOS program itself (file IBMDOS.COM) provides a high-level interface for user programs. It consists of file management routines, data blocking/deblocking for the disk routines, and a variety of built-in functions easily accessible by user programs. (See Appendix D.)

   When these function routines are invoked by a user program, they accept high-level information (for device input/output) via register and control block contents, then (for device operations) translate the requirement into one or more calls to IBMBIO to complete the request.

DOS Initialization

When the system is started (either System Reset or power ON with the DOS diskette in drive A), the boot record is read into memory and given control. It checks the directory to assure that the first two files listed are IBMBIO.COM and IBMDOS.COM, in that order. (An error message is issued if not.) The boot record then reads these two files into memory from absolute diskette sectors (The file allocation table is not used to locate the sectors for these files.), starting at segment X'60', offset 0 (absolute memory address X'600'), jumps to that location (the first byte of IBMBIO.COM).

The beginning of IBMBIO.COM contains a jump to its initialization code, which is located at the high-address end of the program. (This area will later be used as stack space by IBMBIO.COM.) The initialization code determines equipment status, resets the diskette system, initializes the attached devices, and sets the low-numbered interrupt vectors. It then relocates IBMDOS.COM downward to segment X'B0', and calls the first byte of DOS.

As in IBMBIO.COM, offset 0 in DOS contains a jump to its initialization code, which will later be overlaid by a data area and the command processor. DOS initializes its internal working tables, determines the correct memory locations for file allocation table (1 per drive), directory and data buffers, initializes interrupt vectors for interrupts X'20' through X'26' and builds a Program Segment Prefix (See Appendix E.) for COMMAND.COM at the lowest available segment, then returns to IBMBIO.COM.

The last remaining task of initialization is for IBMBIO.COM to load COMMAND.COM at the location set up by DOS initialization. IBMBIO.COM then passes control to the first byte of COMMAND.

The Command Processor

The command processor supplied with DOS (file COMMAND.COM) consists of three distinctly separate parts:

1. A resident portion resides in memory immediately following IBMDOS.COM and its data area. This portion contains routines to process interrupt types X'22' (end address), X'23' (CTRL-BREAK handler), X'24' (critical error handling) and X'27' (end but stay resident), as well as a routine to reload the transient portion if needed. (When a program ends, a checksum methodology determines if the program had caused the transient portion to be overlaid. If so, it is reloaded.) Note that all standard DOS disk error handling is done within this portion of COMMAND. This includes displaying error messages and interpreting the reply of Abort, Retry, or Ignore. (See message "Disk error reading drive x" in Appendix A.)

2. An initialization portion follows the resident portion and is given control during startup. This section contains the AUTOEXEC file processor setup and also the date prompt routine (used if no AUTOEXEC file is found). The initialization portion determines the segment address at which programs can be loaded. It is overlaid by the first program COMMAND loads because it is no longer needed.

3. A transient portion is loaded at the highest end of memory. This is the command processor itself, containing all of the internal command processors, the batch file processor, and a routine to load and execute external commands (files with filename extensions of .COM or .EXE). This portion of COMMAND produces the system prompt (such as A>), reads the command from the keyboard (or batch file) and causes it to be executed. For external commands, it builds a Program Segment Prefix control block immediately following the resident portion of COMMAND, loads the program named in the command into the segment just created, sets the end and CTRL-BREAK exit

address (interrupt vectors X'22' and X'23') to point to the resident portion of COMMAND, then gives control to the loaded program.

> **Note:** Files with an extension of .EXE which are designated to load into high memory are loaded immediately **below** the transient portion of COMMAND to prevent the loading process from overlaying COMMAND itself.

Appendix E contains detailed information describing the conditions in effect when a program is given control by COMMAND.

Replacing the Command Processor

Though the command processor is an important part of DOS, its functions may not be needed in certain environments. Therefore, it has been designed as a user program to allow its replacement. If you decide to replace it with your own command processor:

1. Name your program file COMMAND.COM.

2. The entry conditions are the same as for all .COM programs.

3. Be sure to set the end and CTRL-BREAK exit addresses in the interrupt vectors and in your own Program Segment Prefix to transfer control to your own code.

4. You must provide code to handle (and set the interrupt vectors for) interrupt types X'22' (end address), X'23' (CTRL-BREAK handler), X'24' (critical error handling) and if needed X'27' (end but stay resident). Your COMMAND.COM is also responsible for reading commands from the keyboard and loading and executing programs, if needed.

Available DOS Functions

DOS provides a number of functions to user programs, all available through issuance of a set of interrupt codes. There are routines for keyboard input (with and without echo and CTRL-BREAK detection), console and printer output, constructing file control blocks, memory management, date and time functions, and a variety of diskette and file handling functions. See DOS Interrupts and Function Calls in Appendix D for detailed information.

Diskette/File Management Notes

Through the INT 21 (function call) mechanism, DOS provides methods to create, read, write, rename, and erase files. Files are not necessarily written sequentially on diskette—space is allocated one sector at a time as it is needed, and the first sector available is allocated as the next sector of a file being written. Therefore, if considerable file creation and erasure activity has taken place, newly created files will probably **not** be written in sequential sectors.

However, due to the mapping (chaining) of file sectors via the File Allocation Table, and the fields defined in the File Control Block, any file can be used in either a sequential or random manner. By using the current block and current record fields of the FCB, and the sequential disk read or write functions, you can make the file appear sequential—DOS will do the calculations necessary to locate the proper sectors on the diskette. On the other hand, by using the random record field, and random disk functions, you can cause any record in the file to be accessed **directly**—again, DOS will locate the correct sectors on the diskette for you. Among the most powerful functions are the random block read and write functions, which allow reading or writing a large amount of data with one function call—this is how DOS loads programs. As above, DOS will handle locating the correct sectors on diskette to provide the image of sequential processing—you need not be concerned about the physical location of data on diskette.

The Disk Transfer Area (DTA)

The Disk Transfer Area (also commonly called "buffer") is the memory area DOS will use to contain the data for all disk reads and writes. This area can be at any location within memory, and should be set by your program. (See function call X'1A'.)

Only one DTA can be in effect at a time, so it is the program's responsibility to inform DOS what memory location to use **before** using any disk read or write functions. Once set, DOS continues to use that area for all disk operations until another function call X'1A' is issued to define a new DTA. When a program is given control by COMMAND, a default DTA has already been established at X'80' into the program's Program Segment Prefix, large enough to hold 128 bytes.

Error Trapping

DOS provides a method by which a program can receive control whenever a disk read/write error occurs, or when a bad memory image of the file allocation table is detected. When these events occur, DOS executes an INT X'24' to pass control to the error handler. The default error handler resides in COMMAND.COM, but any program can establish its own by setting the INT X'24' vector to point to the new error handler. DOS provides error information via the registers and provides Abort, Retry, or Ignore support via return codes. (See DOS Interrupts and Function Calls in Appendix D.)

Unlike the end and CTRL-BREAK exit addresses, DOS does not preserve the original contents of the critical error exit address when a program is given control. It is your program's responsibility to preserve the original contents (two words) of the INT X'24' vector prior to setting this vector, and to restore the original contents before ending.

General Guidelines

The following guidelines and tips should assist in developing applications using the DOS disk read and write functions:

1. All disk operations require a properly constructed FCB that the program must supply.

2. Remember to set the Disk Transfer Area address (function X'1A') before performing any reads or writes to a file.

3. All files must be opened (or **created**, in the case of a new file) before being read from or written to. Files which have been written to must also be closed to ensure accurate directory information.

4. A program may define its own logical record size by placing the desired size into the FCB. DOS then uses that value to determine a record's location within the file. If using the "file size" function call, this field **must** be set by the calling program **prior** to the function call. If using the disk read and write routines, the field should be set after opening (or creating) the file but before any read or write functions are used. (Open function sets the field to a default value of 128 bytes.)

5. New files must be created (function call X'16') before they can be written to. This call creates a new directory entry **and** opens the file.

6. If the amount of data being transferred is less than one sector (512 bytes), DOS will "buffer" the data for the requesting program in an internal buffer within IBMDOS.COM. Because there is only one disk buffer, performing less-than-sector-size operations in a random manner or against multiple files concurrently causes DOS to frequently change the contents of the buffer. If such operations are in output mode, this forces DOS to write a partially full sector to make the buffer available for any other diskette operation.

Subsequently, the partially full sector would have to be re-read before further data could be written to the file. This is called "thrashing" and can be very time-consuming. To remedy this situation, use of the Random block read and write routines is recommended, with a data transfer size as large as possible. (An entire file can be read this way, provided enough memory exists.) This method bypasses the "buffering" described above, by reading or writing directly to or from the DTA for as much of the data as possible. If the file size is not a multiple of 512 bytes, only the last portion of the file (the portion past the last 512-byte multiple) is buffered by DOS.

Example of Using DOS Functions

This example illustrates the steps necessary for a program named TEST.COM to:

1. Create a new file named FILE1.

2. Load and execute a second program named PGM1.COM from the diskette in drive B.

The program is in a file named TEST.COM, and was invoked from the keyboard by the command TEST FILE1 B:PGM1.COM.

When the program (TEST) receives control, the Program Segment Prefix has been set up as shown in Appendix E. The end and CTRL-BREAK exit addresses in the Program Segment Prefix are the ones which the host (calling program) had established and should not be modified—they are restored to interrupt X'22' and X'23' vectors when this program ends. The FCBs at X'5C' and X'6C' are formatted to contain file names of FILE1 and PGM1.COM, respectively—the first FCB reflects the default drive and the second, drive B. The default DTA is set to X'80' into the segment (the unformatted parameter area of the Program Segment Prefix).

To Create File FILE1

Because it is known that the data in the FCB at X'6C' is needed to load and execute the program whose name it contains in a subsequent step, that FCB must be preserved; opening the FCB at X'5C' would cause it to be overlayed. The program should:

1. Copy the FCB at X'6C' to an area within itself.

2. Using the FCB at X'5C'; call function X'11' to be sure FILE1 does not already exist—if it did exist, it would be overwritten by this program.

3. Assuming it did not exist, create the file (function call X'16')—the file is now open.

4. Set the FCB current record and random record fields to zero, and the record size field to the desired size.

5. Build the memory image of the file's data.

6. Set the DTA to point to the memory image (function call X'1A').

7. Use the sequential write (X'15'), random write (X'22'), or random block write (X'28') calls to write the file, ensuring the FCB fields and DTA are set properly for each call. In the case of call X'28' (the preferred method), the entire file can be written with one call by setting CX to the number of records to be written (in terms of the FCB record size field).

8. Close the FCB at X'5C'—the directory and file allocation table are updated, and any partial data in DOS's disk buffer (if it were performing blocking) are written to disk.

To Load and Execute Program PGM1.COM from Drive B.

Assume that the current program (TEST) wished to control the action taken if CTRL-BREAK is entered. (Until now, the CTRL-BREAK address still pointed to COMMAND.COM, which would end program TEST if CTRL-BREAK were pressed.)

TEST should:

1. Set the end and CTRL-BREAK exit vectors (call X'25') to point to code within itself (the end address is where the program to be loaded will return when it ends).

2. Determine where PGM1.COM should reside in memory and set up a segment for it, including a Program Segment Prefix (call X'26'). This copies the end and CTRL-BREAK exit addresses just set into the new segment's Program Segment Prefix.

3. Set the DTA to offset X'100' into the just-created segment. (Be sure the DS register contains the correct segment address.) This is the offset at which PGM1.COM will be loaded.

4. Open the FCB that had been copied earlier (for PGM1.COM). The FCB file size field will be filled in by open to a default value of 128 bytes.

5. Set the FCB record size field to the desired size. (Setting it to 1 is very useful in this case.)

6. Set the CX register to the number of records (based on the record size field) to read. If the record size was set to 1, then the number of records to read does not have to be computed—it can be obtained directly from the FCB file size field. In any case, if the product of the record size field and contents of the CX register are equal to or greater than the file size, then the entire file is read in the following step.

7. Read the file, using the Random Block Read function (call X'27'), into the new segment at offset X'100'. (See step 3 above.) There is no need to close the file since it was not written to.

8. Prepare the DS, ES, SS and SP registers for the loaded program and push a word of zeros on the top of its stack.

9. Set the DTA to offset X'80' into the new segment.

10. Give control to the loaded program. (An intersegment jump is ideal, since it does not use stack space.) When the called program ends via INT X'20', DOS restores interrupt vectors X'22' and X'23' from the values in the ending program's Program Segment Prefix (the values established in step 1) and pass control to the end exit address. TEST is now back in control, and can itself issue an INT X'20', which will cause its caller (COMMAND.COM) to regain control.

> **Note:** The example above was simplified by not discussing the checking of return codes from the function calls. Nearly all function calls **do** return exception or error indications, which should be checked by the calling program.

Appendix C. DOS Diskette Allocation

The single-sided 40-track (0-39) diskettes have eight sectors per track, with 512 bytes per sector.

DOS allocates space on the diskette as follows:

Track 0 sector 1	Boot record written by the FORMAT command.
Track 0 sectors 2-3	Two copies of the File Allocation Table (FAT), one in each sector.
Track 0 sectors 4-7	Directory.
Track 0 sector 8 to Track 39 sector 8	Data area.

Detailed descriptions of the directory and File Allocation Table are presented in this appendix.

Files are not necessarily written sequentially on the diskette. Diskette space for a file in the data area is allocated one sector at a time, skipping over sectors already allocated. The first free sector found will be the next sector allocated, regardless of its location on the diskette. This permits the most efficient utilization of diskette space because sectors made available by erasing files can be allocated for new files. (See the description of the File Allocation Table.)

The minimum allocation unit is one sector; therefore, all files begin on a sector boundary.

> **Note:** If the diskette contains a copy of DOS, it is placed in the data area as follows:
>
> IBMBIO.COM – track 0 sector 8 through track 1 sector 3
> IBMDOS.COM – track 1 sector 4 through track 2 sector 8

These two programs must reside at the specific locations indicated so that the boot record can successfully load them when the system is started.

DOS Diskette Directory

FORMAT builds the directory for each diskette on track 0 sectors 4-7, a total of 2048 bytes. The directory has room for 64 entries, each 32 bytes long. Each directory entry is formatted as follows. (Byte offsets are in decimal.)

0-7 Filename. (X'E5' in byte 0 means this directory entry is not used.)

8-10 Filename extension.

11 File attribute. Contents can be X'02' for a hidden file and X'04' for a system file. (Both files are excluded from all directory searches unless an extended FCB with the appropriate attribute byte is used.) For all other files this byte contains X'00'. A file can be designated as hidden when it is created.

12-23 Reserved.

24-25 Date the file was created or last updated. The mm/dd/yy are mapped in the bits as follows:

```
<          25          ><          24          >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
y  y  y  y  y  y  y m m m m d d d d d
```

where:

mm is 1-12
dd is 1-31
yy is 0-119 (1980-2099)

26-27 Starting sector; the relative sector number of the first block in the file. (For file allocation purposes only, relative sector numbers start at 000 with track 0 sector 6. This is in contrast with DEBUG and the absolute disk read/write routines, interrupts X'25' and X'26', which number relative sectors from the beginning of the diskette.)

Note: Relative sectors 000 and 001 are the last two sectors of the directory. Therefore, the data area (track 0 sector 8) starts with relative sector 002.

The relative sector number is stored with the least significant byte first.

To calculate the absolute track/sector:

1. Add 5 to the relative sector number (to include the five sectors before relative sector 000).

2. Divide by 8 (8 sectors per track).

3. The quotient is the track number.

4. The remainder is one less than the sector number.

28-31 File size in bytes. The first word contains the low-order part of the size. Both words are stored with the least significant byte first.

Note: If the diskette was formatted with the /S option (FORMAT command), the first three files in the directory are IBMBIO.COM, IBMDOS.COM, and COMMAND.COM. A special file (BADTRACK) is present if defective tracks were found during the disk initialization process. Any defective tracks are allocated to this file to prevent them from being allocated to a user file.

The first two system files are placed on specific sectors. Because they occupy specific sectors and cannot be moved, they are protected from accidental erasure or destruction by being excluded from all directory searches. (See the file-attribute byte in the directory.)

DOS File Allocation Table

The File Allocation Table (FAT) is used by DOS to allocate diskette space for a file, one sector at a time.

The FAT consists of a 12-bit entry (1.5 bytes) for each sector, starting with track 0 sector 6 and continuing through track 39 sector 8.

Note that the first two FAT entries map the last two sectors of the directory; so, these FAT entries contain indicators of the size and format of the directory. (The last two sectors of the directory are track 0 sectors 6-7; for allocation purposes, they are relative sectors 000 and 001.)

The third FAT entry begins the mapping of the data area starting with track 0 sector 8 (relative sector 002).

Each entry contains three hexadecimal characters:

000 if the sector is unused and available, or

FFF to indicate the last sector of a file, or

XXX any other hexadecimal characters that are the relative sector number of the **next sector** in the file. The relative sector number of the first sector in the file is kept in the file's directory entry.

A copy of the FAT for the last used diskette in each drive is kept in memory (See the DOS Memory Map.) and is written to track 0 sectors 2 and 3 whenever the status of diskette space changes.

How to Use the File Allocation Table

Obtain the **starting sector** of the file from the directory entry. Calculate the absolute track/sector as follows:

1. Add 5 to the relative sector number (to include the 5 sectors before relative sector 000).

2. Divide by 8 (8 sectors per track).

3. The quotient is the track number.

4. The remainder is one less than the sector number.

Now, to locate the **next sector** of the file:

1. Multiply the relative sector number just used by 1.5 (each FAT entry is 1.5 bytes long).

2. The whole part of the product is an offset into the FAT, pointing to the entry that maps the sector just used. That entry contains the relative sector number of the next sector of the file.

3. Use a MOV instruction to move the word at the calculated FAT offset into a register.

4. If the last relative sector used was an even number, keep the low-order 12 bits of the register; otherwise, keep the high-order 12 bits.

5. If the resultant 12 bits are all 1's (X'FFF'), there are no more sectors in the file. Otherwise, the 12 bits contain the relative sector number of the next sector in the file.

Appendix D.
DOS Interrupts and Function Calls

Interrupts

DOS reserves interrupt types X'20' to X'3F' for its use. This means absolute memory locations X'80' to X'FF' are the transfer address locations reserved by DOS. The defined interrupts are as follows with all values in hexadecimal.

20 Program end. Issuing Interrupt X'20' is the normal way to exit from a program. This vector transfers to the logic in DOS for restoration of the end and CTRL-BREAK exit addresses to the values they had on entry to the program. All file buffers are flushed to diskette. All files changed in length should be closed (See function call X'10'.) prior to issuing this interrupt. If the changed file is not closed, its length is not recorded correctly in the directory.

> **Note:** Every program must ensure that the CS register contains the segment address of its Program Segment Prefix control block prior to issuing INT X'20'.

21 Function request. See "Function Calls", below.

22 End address. The address represented by this interrupt is the address to which control transfers when the program ends. This address is copied into the program's Program Segment Prefix at the time the segment is created. If a program wishes to execute a second program it must set the end address prior to creating the segment into which the new program will be loaded. Otherwise, when the second program executes, its ending would cause transfer to its host's end address. This address, as well as the CTRL-BREAK address below, may be set via DOS function call X'25'.

23 CTRL-BREAK exit address. If the user enters CTRL-BREAK **during keyboard input or display output**, an interrupt type X'23' is executed. If the CTRL-BREAK routine saves all registers, it may end with a return-from-interrupt instruction (IRET) to continue program execution. If the CTRL-BREAK interrupts functions 9 or 10, buffered I/O, then a backslash, carriage-return, and linefeed are output. If execution is then continued with an IRET, I/O continues from the start of the line. When the interrupt occurs, all registers are set to the value they had when the original function call to DOS was made. There are no restrictions on what the CTRL-BREAK handler is allowed to do, including DOS function calls, as long as the registers are unchanged if IRET is used.

If the program creates a new segment and loads in a second program which itself changes the CTRL-BREAK address, the end of the second program and return to the first causes the CTRL-BREAK address to be restored to the value it had before execution of the second program. (It is restored from the second program's Program Segment Prefix.)

24 Critical error handler vector. When a critical error occurs within DOS, control is transferred with an INT 24H. On entry to the error handler, AH will have its bit 7=0 (high-order bit) if the error was a hard disk error (probably the most common occurrence), bit 7=1 if not.

Currently, the only error possible when AH bit 7=1 is a bad memory image of the file allocation table.

If it is a hard disk error, AH bits 0-2 indicate the affected disk area and whether it was a read or write operation, as follows.

Bit 0=0 if read operation,
1 if write operation.

Bits 2-1 (affected disk area)

0 0 DOS area (system files)
0 1 file allocation table
1 0 directory
1 1 data area

The registers will be set up for a retry operation, and an error code will be in the lower half of the DI register with the upper half undefined. These are the error codes.

Error code	Description
0	Attempt to write on write-protected diskette
2	Drive not ready
4	Data error
6	Seek error
8	Sector not found
A	Write fault
C	General disk failure

The user stack will be in effect, and will contain the following from top to bottom.

IP CS FLAGS	DOS registers from issuing INT X'24'
AX BX CX DX SI DI BP DS ES	User registers at the time of original INT X'21' request
IP CS FLAGS	From the original interrupt X'21' from the user to DOS

The registers are set such that if an IRET is executed, DOS will respond according to (AL) as follows:

(AL)=0 ignore the error.

=1 retry the operation. (If this option is used, then the stack, SP, SS, DS, BX, CX, and DX must **not** be modified.)

=2 end the program.

Notes:

1. Before giving this routine control for disk errors, DOS performs three retries.

2. For disk errors, this exit is taken only for errors occurring during an INT X'21' function call. It is not used for INT X'25' or X'26'.

3. If you set this vector, be sure to preserve the original contents and restore them before your program ends. Otherwise, unpredictable results will occur.

4. If you decide to handle the error yourself without returning to DOS, be sure to clear the stack of the first and last three words and restore your registers from the stack. Also, this routine should enable interrupts because it was entered with interrupts disabled.

25 Absolute disk read. This transfers control directly to the DOS BIOS. Upon return, the original flags are still on the stack (put there by the INT instruction). This is necessary because return information is passed back in the current flags.

Be sure to pop the stack to prevent uncontrolled growth. For this entry point "records" and "sectors" are the same size. The request is as follows:

(AL)	Drive number (For example, 0=A or 1=B)
(CX)	Number of sectors to read
(DX)	Beginning logical record number
(DS:BX)	Transfer address

The number of records specified is transferred between the given drive and the transfer address. "Logical record numbers" are obtained by numbering each sector sequentially starting from zero and continuing across track boundaries. For example, logical record number 0 is track 0 sector 1, whereas logical record number X'12' is track 2 sector 3.

All registers except the segment registers are destroyed by this call. If the transfer was successful, the carry flag (CF) will be zero. If the transfer was not successful, CF=1 and (AL) will indicate the error as follows:

X'80'	Attachment failed to respond
X'40'	SEEK operation failed
X'20'	Controller failure
X'10'	Bad CRC on diskette read
X'08'	DMA overrun on operation
X'04'	Requested sector not found
X'03'	Write attempt on write-protected diskette
X'02'	Address mark not found

26 Absolute disk write. This vector is the counterpart of interrupt 25 above. Except for the fact that this is a write, the description above applies.

27 End but stay resident. This vector is used by programs that are to remain resident when COMMAND regains control. After initializing itself, the program must set DX to its last address plus one in the segment in which it is executing (the offset at which COMMAND can load other

programs), then execute an INT 27H. COMMAND then considers the program as an extension of DOS, so the program is not overlaid when other programs are executed. This concept is very useful for loading programs such as user-written interrupt handlers which must remain resident.

> **Note:** This interrupt must **not** be used by .EXE programs which are loaded into the high end of memory.

Function Calls

DOS functions are called by placing a function number in the AH register, supplying additional information in other registers as necessary for the specific function, then executing an interrupt type X'21'. When DOS takes control, it switches to an internal stack. User registers are preserved unless information is passed back to the requester as indicated in the specific requests. The user stack needs to be sufficient to accommodate the interrupt system. It is recommended that it be X'80' in addition to the user needs.

There is an additional mechanism provided for pre-existing programs that were written with different calling conventions. The function number is placed in the CL register, other registers are set according to the function specification, and an intrasegment call is made to location 5 in the current code segment. That location contains a long call to the DOS function dispatcher. Register AX is always destroyed if this mechanism is used; otherwise, it is the same as normal function calls. This method is valid only for function calls 0-24 (hexadecimal).

The functions are as follows with all values in hexadecimal.

0 Program end. The end and CTRL-BREAK exit addresses are restored to the values they had on entry to the ending program, from the values saved in the Program Segment Prefix. All file buffers are flushed, but any files which have been changed in length but not closed will not be recorded properly in the directory. Control transfers to the end address.

Note: This call performs exactly the same function as INT 20H. It is the program's responsibility to ensure that the CS register contains the segment address of its Program Segment Prefix control block prior to calling this function.

1 Keyboard input. Waits for a character to be typed at the keyboard (unless one is ready), then echos the character to the display and returns it in AL. The character is checked for a CTRL-BREAK. If CTRL-BREAK is detected, an interrupt X'23' is executed.

Note: For functions 1, 6, 7, and 8, extended ASCII codes will require two function calls. (See the IBM Personal Computer BASIC manual for a description of the extended ASCII codes.) The first call returns 00 as an indicator that the next call will return an extended code.

2 Display output. The character in DL is output to the display. The backspace character results in moving the cursor left one position, writing a space at this position and remaining there. If a CTRL-BREAK is detected after the output, an interrupt X'23' is executed.

3 Auxiliary (Asynchronous Communications Adapter) input. Waits for a character from the auxiliary input device, then returns that character in AL.

Notes:

1. Auxiliary support is unbuffered and non-interrupt driven.

2. At start-up, DOS initializes the first auxiliary port to 2400 baud, no parity, one stop bit, and 8-bit word.

3. The auxiliary function calls (3 and 4) do not return status or error codes. For greater control, it is recommended that the ROM BIOS routine (INT X'14') be used.

4 Auxiliary (Asynchronous Communications Adapter) output. The character in DL is output to the first auxiliary device.

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

6 Direct console I/O. If DL is X'FF', AL returns with a keyboard input character if one is ready, otherwise 00. If DL is not X'FF', then DL is assumed to have a valid character which is output to the display. This function does not check for CTRL-BREAK.

7 Direct console input without echo. Waits for a character to be typed at the keyboard (unless one is ready), then returns the character in AL. As with function 6, no checks are made on the character.

8 Console input without echo. This function is identical to function 1, except the key is not echoed.

9 Print string. On entry, DS:DX must point to a character string in memory terminated by a "$" (X'24'). Each character in the string will be output to the display in the same form as function 2.

A Buffered keyboard 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 keyboard and placed in the buffer beginning at the third byte. Reading the keyboard and filling the buffer continues until ENTER is pressed. If the buffer fills to one less than the maximum number of characters it can hold, then each additional character typed is ignored and causes the bell to ring, until ENTER is pressed. The second byte of the buffer is set to the number of characters received excluding the carriage return (X'0D'), which is always the last character. Editing of this buffer is described in Chapter 1.

B Check keyboard status. If a character is available from the keyboard, AL will be X'FF'. Otherwise, AL will be 00. If a CTRL-BREAK is detected, an interrupt type X'23' is executed.

C Clear keyboard buffer and invoke a keyboard input function. Clear the keyboard buffer of any pre-typed characters, then execute the function number in AL (only 1, 6, 7, 8, and A are allowed). This forces the system to wait until a character is typed.

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

E Select disk. The drive specified in DL (0=A, 1=B) is selected (if valid) as the default drive. The number of drives is returned in AL. (A value of 2 is returned in a single-drive system to be consistent with the philosophy of thinking of the system as having logical drives A and B. BIOS equipment determination (INT 11H) can be used as an alternative method, returning the actual number of physical drives.)

F Open file. On entry, DS:DX point to an unopened file control block (FCB). The directory is searched for the named file and AL returns X'FF' if it is not found. If it is found, AL returns 00 and the FCB is filled as follows:

If the drive code was 0 (default drive), it is changed to actual drive used (1=A, 2=B). This allows changing the default drive without interfering with subsequent operations on this file. The current block field (FCB bytes C-D) is set to zero. The size of the record to be worked with (FCB bytes E-F) is set to the system default of X'80'. The size of the file and the date are set in the FCB from information obtained from the directory.

It is your responsibility to set the record size (FCB bytes E-F) to the size you wish to think of the file in terms of, if the default X'80' is insufficient. It is also your responsibility to set the random record field and/or current record field. These actions should be done after open but before any disk operations are requested.

10 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 diskette was changed and AL returns X'FF'. Otherwise, the directory is updated to reflect the status in the FCB and AL returns 00.

11 Search for the first entry. On entry, DS:DX point to an unopened FCB. The disk directory is searched for the first matching filename (name could have "?"s indicating any letter matches) and if none is found AL returns X'FF'. Otherwise, AL returns 00 and the locations at the disk transfer address are set as follows.

If the FCB provided for searching was an extended FCB, then the first byte at the disk transfer address is set to X'FF', followed by five bytes of zeros, then the attribute byte from the search FCB, then the drive number used (1=A, 2=B), then the 32 bytes of the directory entry. Thus, the disk transfer address contains a valid unopened extended FCB with the same search attributes as the search FCB.

If the FCB provided for searching was a normal FCB, then the first byte is set to the drive number used (1=A, 2=B), and the next 32 bytes contain the matching directory entry. Thus, the disk transfer address contains a valid unopened normal FCB.

See Appendix C on DOS Diskette Allocation for the format of directory entries.

12 Search for the next entry. After function 11 has been called and found a match, function 12 may be called to find the next match to an ambiguous request ("?"s in the search filename). Both inputs and outputs are the same as function 11. The reserved area of the FCB keeps information necessary for continuing the search, so no disk operations may be performed with this FCB between a previous function 11 or 12 call and this one.

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

14 Sequential read. On entry, DS:DX point to an opened FCB. The record addressed by the current block (FCB bytes C-D) and the current record (FCB byte 1F) is loaded at the disk transfer address, then the record address is incremented. (The length of the record is determined by the FCB record size field.) If end-of-file is encountered, AL returns either 01 or 03. A return of 01 indicates no data in the record, 03 indicates a partial record is read and filled out with zeros. A return of 02 means there was not enough space in the disk transfer segment to read one record; so, the transfer was ended. AL returns 00 if the transfer was completed successfully.

15 Sequential write. On entry, DS:DX point to an opened FCB. The record addressed by the current block and current record fields (size determined by the FCB record size field) is written from the disk transfer address (or, in the case of records less than sector sizes, is buffered up for an eventual write when a sector's worth of data is accumulated). The record address is then incremented. If the diskette is full AL returns 01. A return of 02 means there was not enough space in the disk transfer segment to write one record; so, the transfer was ended. AL returns 00 if the transfer was completed successfully.

16 Create file. On entry, DS:DX point to an unopened FCB. The disk directory is searched for a matching entry and if found, it is re-used. If no match was found, the directory is searched for an empty entry, and AL returns FF if none is found. Otherwise, the entry is initialized to a zero-length file, the file is opened (See function F.), and AL returns 00.

The file may be marked "hidden" during its creation by using an extended FCB containing the appropriate attribute byte.

17 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+X'11') in what is normally a reserved area. Every matching occurrence of the first name is changed to the second (with the restriction that two files cannot have the same name and extension). If "?"s appear in the second name, then the corresponding positions in the original name will be unchanged. AL returns FF if no match was found or if an attempt was made to rename to a filename that already existed, otherwise 00.

18 Not used.

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

1A Set disk transfer address. The disk transfer address is set to DS:DX. DOS does not allow disk transfers to wrap around within the segment, or overflow into the next segment.

1B Allocation table address. On return, DS:BX point to the file allocation table for the current drive, DX has the number of allocation units, AL has the number of records per allocation unit, and CX has the size of the physical sector.

1C Not used.

1D Not used.

1E Not used.

1F Not used.

20 Not used.

21 Random read. On entry, DS:DX point to an opened FCB. The current block and current record fields are set to agree with the random record field, then the record addressed by these fields is read into memory at the current disk transfer address. If end-of-file is encountered, AL returns either 01 or 03. If 01 is returned, no more data is available. If 03 is returned, a partial record is available filled out with zeros. A return of 02 means there was not enough space in the disk transfer segment to read one record; so, the transfer was ended. AL returns 00 if the transfer was completed successfully.

22 Random write. On entry, DS:DX point to an opened FCB. The current block and current record fields are set to agree with the random record field, then the record addressed by these fields is written (or in the case of records not the same as sector sizes–buffered) from the disk transfer address. If the diskette is full AL returns 01. A return of 02 means there was not enough space in the disk transfer segment to write one record; so, the transfer was ended. AL returned 00 if the transfer was completed successfully.

23 File size. On entry, DS:DX point to an unopened FCB. The diskette directory is searched for the first matching entry and if none is found, AL returns FF. Otherwise, the random record field is set to the number of records in the file (in terms of the record size field rounded up) and AL returns 00.

> **Note:** Be sure to set the FCB record size field before using this function call; otherwise, erroneous information will be returned.

24 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 record fields.

25 Set interrupt vector. The interrupt vector table for the interrupt type specified in AL is set to the 4-byte address contained in DS:DX.

26 Create a new program segment. On entry, DX has a segment number at which to set up a new program segment. The entire X'100' area at location zero in the current program segment is copied into location zero in the new program segment. The memory size information at location 6 in the new segment is updated and the current end and CTRL-BREAK exit addresses (from interrupt vector table entries for interrupt types 22 and 23) are saved in the new program segment starting at X'0A'. They are restored from this area when the program ends.

27 Random block read. On entry, DS:DX point to an opened FCB, and CX contains a record count that must not be zero. The specified number of records (in terms of the record size field) is 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, AL returns either 01 or 03. A return of 01 indicates end-of-file and the last record is complete, a 03 indicates the last record is a partial record. If wrap-around above address X'FFFF' in the disk transfer segment would have occurred, 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 field and the current block/record fields are set to address the next record (the first record not read).

28 Random block write. Essentially the same as function 27 above, except for writing and a write-protect check. If there is insufficient space on the disk, AL returns 01 and no records are written. If CX is zero upon entry, no records are written, but the file is set to the length specified by the random record field, whether longer or shorter than the current file size. (Allocation units are released or allocated as appropriate.)

29 Parse filename. On entry, DS:SI point to a command line to parse, and ES:DI point to a portion of memory to be filled with an unopened FCB. If AL=1 on entry, then leading separators are scanned off the command line at DS:SI. If AL=0, then no scan-off of leading separators takes place.

Filename separators include the following characters : . ; , = + / " [ ] plus TAB and SPACE. Filename terminators include all of these characters plus any control characters.

The command line is parsed for a filename of the form d:filename.ext, and if found, a corresponding unopened FCB is created at ES:DI. If no drive specifier is present, the default drive is assumed. If no extension is present, it is assumed to be all blanks. If the character "*" appears in the filename or extension, then it and all remaining characters in the name or extension are set to ?.

If either ? or * appears in the filename or extension, AL returns 01; if the drive specifier is invalid AL returns FF; otherwise 00.

DS:SI will return pointing to the first character after the filename and ES:DI will point to the first byte of the formatted FCB. If no valid filename is present, ES:DI+1 will contain a blank.

2A Get date. Returns date in CX:DX. CX has the year (1980-2099 in binary), DH has the month (1-Jan, 2-Feb, etc) and DL has the day. If the time-of-day clock rolls over to the next day, the date is adjusted accordingly, taking into account the number of days in each month and leap years.

2B Set date. On entry, CX:DX must have a valid date in the same format as returned by function 2A, above. If the date is indeed valid and the set operation is successful, AL returns 00. If the date is not valid, AL returns FF.

2C Get time. Returns with time-of-day in CX:DX. Time is actually represented as four 8-bit binary quantities as follows. CH has the hours (0-23), CL has minutes (0-59), DH has seconds (0-59), DL has 1/100 seconds (0-99). This format is readily converted to a printable form yet can also be used for calculations, such as subtracting one time value from another.

2D Set time. On entry, CX:DX has time in the same format as returned by function 2C, above. If any component of the time is not valid, the set operation is ended and AL returns FF. If the time is valid, AL returns 00.

Appendix E.
DOS Control Blocks and Work Areas

DOS Memory Map

0000:0000	Interrupt vector table
0040:0000	ROM communication area
0050:0000	DOS communication area
0060:0000	IBMBIO.COM—DOS interface to ROM I/O routines
00B0:0000	IBMDOS.COM—DOS interrupt handlers, service routines (INT 21 functions)
	Directory buffer
	Disk buffer
	Drive parameter block/file allocation table (one per drive)
XXXX:0000	Resident portion of COMMAND.COM—Interrupt handlers for interrupts X'22' (end), X'23' (CTRL-BREAK), X'24' (critical error), X'27' (end but stay resident), and code to reload the transient portion.
XXXX:0000	External command or utility—(.COM or EXE file)
XXXX:0000	User stack for .COM files (256 bytes)
XXXX:0000	Transient portion of COMMAND.COM—Command interpreter, internal commands, external command processor, batch processor.

Notes:

1. Memory map addresses are in segment:offset format. For example, 0060:0000 is absolute address X'0600'.

2. The DOS Communication Area is used as follows:

   0050:0000 Print screen status flag store

   - **0** Print screen not active or successful print screen operation
   - **1** Print screen in progress
   - **255** Error encountered during print screen operation

   0050:0004 Single-drive mode status byte

   - **0** Diskette for drive A: was last used
   - **1** Diskette for drive B: was last used

DOS Program Segment

When you enter an external command, the COMMAND processor determines the lowest available address (immediately after the resident portion of COMMAND.COM) to use as the start of available memory for the program invoked by the external command. This area is called the Program Segment.

At offset 0 within the Program Segment, COMMAND builds the Program Segment Prefix control block. (See below.) COMMAND loads the program at offset X'100' and gives it control. (.EXE files can be loaded into high memory just below the transient portion of COMMAND.COM, but the Program Segment Prefix will still be in low memory.)

The program returns to COMMAND by a jump to offset 0 in the Program Segment Prefix (The instruction INT 20 is the first item in the control block.), by issuing an INT 20, or by issuing an INT 21 with register AH=0.

> **Note:** It is the responsibility of all programs to ensure that the CS register contains the segment address of the Program Segment Prefix when ending via any of these methods.

All three methods result in an INT 20 being issued, which transfers control to the resident portion of COMMAND.COM. It restores interrupt vectors X'22' and X'23' (end and CTRL-BREAK exit addresses) from the values saved in the Program Segment Prefix of the ending program. Control is then given to the end address. (If this is a program returning to COMMAND, control transfers to its transient portion.) If a batch file was in process, it is continued; otherwise, COMMAND issues the system prompt and waits for the next command to be entered from the keyboard.

When a program receives control, the following conditions are in effect.

For all programs:

- Disk transfer address (DTA) is set to X'80' (default DTA in the Program Segment Prefix).
- File control blocks at X'5C' and X'6C' are formatted from the first two parameters entered when the command was invoked.
- Unformatted parameter area at X'81' contains all the characters entered after the command name (including leading and embedded delimiters), with X'80' set to the number of characters.
- Offset 6 (one word) contains the number of bytes available in the segment. If the resident portion of COMMAND.COM is within the segment, this value is reduced by its size.

- Register AX reflects the validity of drive specifiers entered with the first two parameters as follows:
  - AL=FF if the first parameter contained an invalid drive specifier (otherwise AL=00)
  - AH=FF if the second parameter contained an invalid drive specifier (otherwise AH=00)

For .COM programs:

- All four segment registers contain the segment address of the Program Segment Prefix control block.

- The Instruction Pointer (IP) is set to X'100'.

- SP register is set to the end of the program's segment, or the bottom of the transient portion of COMMAND.COM, whichever is lower. The segment size at offset 6 is reduced by X'100' to allow for a stack of that size.

- A word of zeros is placed on the top of the stack.

For .EXE programs:

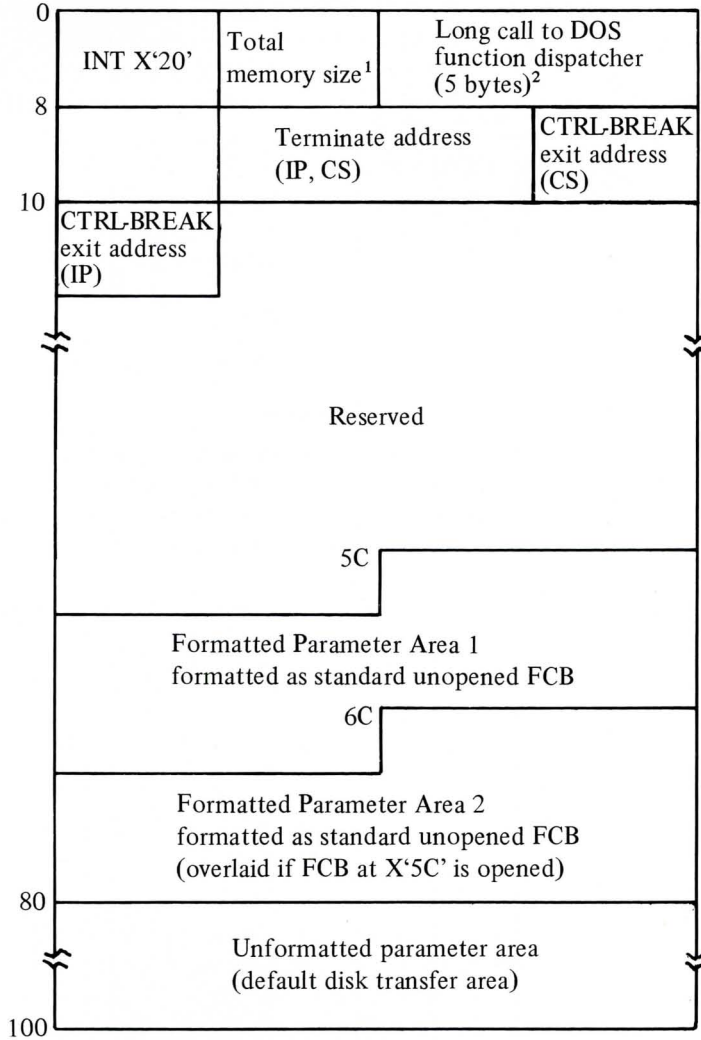
- DS and ES registers are set to point to the Program Segment Prefix. (See below.)

- CS, IP, SS, and SP registers are set to the values passed by the linker.

The Program Segment Prefix (with offsets in hexadecimal) is formatted as follows:

PROGRAM SEGMENT PREFIX

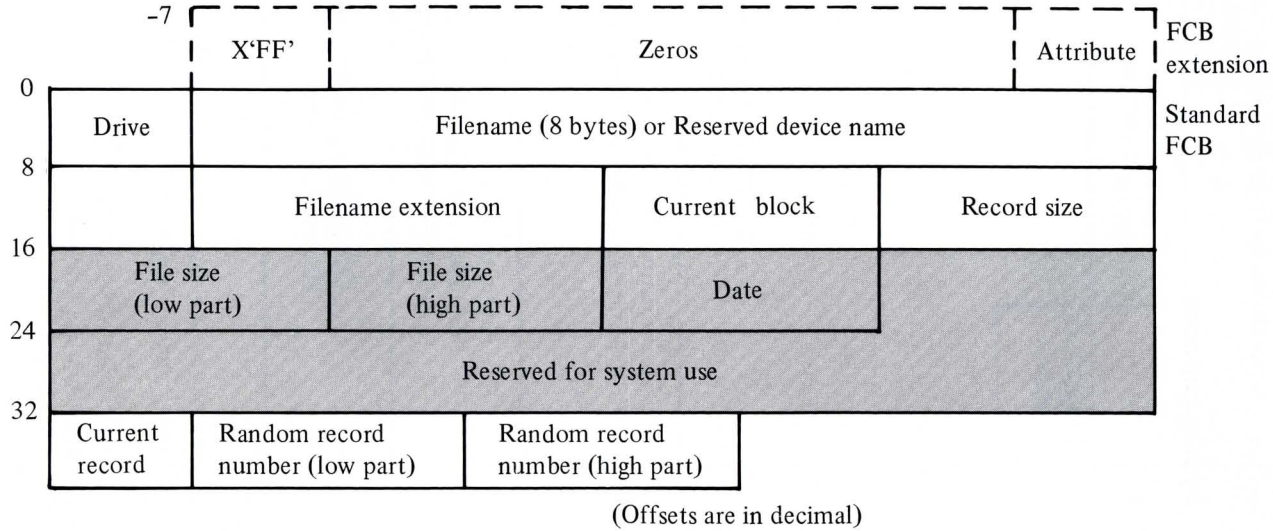
(offsets in hex)



1. Memory size is in segment (paragraph) form (for example, X'1000' would represent 64K).

2. The word at offset 6 contains the number of bytes available in the segment.

FILE CONTROL BLOCK



(Offsets are in decimal)

Unshaded areas must be filled in by the using program.

Shaded areas are filled in by DOS and must not be modified.

Standard File Control Block

The standard File Control Block (FCB) is defined as follows, with the offsets in decimal.

Byte	Function
0	Drive number. For example, **Before open:** 0–default drive 1–drive A 2–drive B **After open:** 1–drive A 2–drive B (A 0 is replaced by the actual drive number during open.)
1-8	Filename, left-justified with trailing blanks. If a reserved device name is placed here (such as LPT1), do not include the optional colon.
9-11	Filename extension, left-justified with trailing blanks (can be all blanks).
12-13	Current block number relative to the beginning of the file, starting with zero (set to zero by the open function call). A block consists of 128 records, each of the size specified in the logical record size field. The current block number is used with the current record field (below) for sequential reads and writes.
14-15	Logical record size in bytes. Set to X'80' by the open function call. If this is not correct, you must set the value because DOS uses it to determine the proper locations in the file for all disk reads and writes.
16-19	File size in bytes. In this 2-word field, the first word is the low-order part of the size.

20-21 Date the file was created or last updated. The mm/dd/yy are mapped in the bits as follows:

```
<          21          >< 20 >
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
y  y  y  y  y  y  y m m m m d d d d d
```

where:

mm is 1-12
dd is 1-31
yy is 0-119 (1980-2099)

22-31 Reserved for system use.

32 Current relative record number (0-127) within the current block. (See above.) You must set this field before doing **sequential** read/write operations to the diskette. (This field is not initialized by the open function call.)

33-36 Relative record number relative to the beginning of the file, starting with zero. You must set this field before doing **random** read/write operations to the diskette. (This field is not initialized by the open function call.)

If the record size is less than 64 bytes, both words are used. Otherwise, only the first three bytes are used. Note that if you use the File Control Block at X'5C' in the program segment, the last byte of the FCB overlaps the first byte of the unformatted parameter area.

Notes:

1. Bytes 0-15 and 32-36 must be set by the user program. Bytes 16-31 are set by DOS and must not be changed by user programs.

2. All word fields are stored with the least significant byte first. For example, a record length of 128 is stored as X'80' at offset 14, and X'00' at offset 15.

Extended File Control Block

The extended File Control Block is used to create or search for files in the diskette directory that have special attributes.

It adds a 7-byte prefix to the FCB, formatted as follows:

Byte	Function
FCB-7	Flag byte containing X'FF' to indicate an extended FCB.
FCB-6 to FCB-2	Reserved.
FCB-1	Attribute byte to include hidden files (X'02') or system files (X'04') in directory searches. IBMBIO.COM, IBMDOS.COM, and BADTRACK are considered to be system files. There are no "hidden" files supplied on the DOS diskette. This function is present to allow applications to define their own files as "hidden", and thereby exclude them from directory searches. This prevents them from being accidentally erased or overwritten by a COPY command.

Any reference in the DOS Function Calls (See Appendix D.) to an FCB, whether opened or unopened, may use either a normal or extended FCB. (If using an extended FCB, the appropriate register should be set to the first byte of the prefix, rather than the drive-number field.)