IBM

Personal Computer
Computer Language
Series

Disk Operating System

6172220

IBM

Personal Computer
Computer Language
Series

Disk Operating System

First Edition Revised (January 1982)

Changes are periodically made to the information herein; these changes will be incorporated in new editions of this publication.

Products are not stocked at the address below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM Personal Computer Dealer.

A Product Comment Form is provided at the back of this publication. If this form has been removed, address comment to: IBM Corp., Personal Computer, P.O. Box 1328-C, Boca Raton, Florida 33432. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligations whatever.

CONTENTS

Preface vii
Prerequisite Publication vii
Organization of this manual vii
Data Security viii

CHAPTER 1. INTRODUCTION 1-1
What is DOS? 1-3
What are the parts of DOS? 1-3
A Few Words about Files 1-5
What is a file? 1-5
What can I name my files? 1-5
How many files can I have? 1-6
How does DOS keep track of my files? . . . 1-6
Why is this information important to me? . . 1-6
Formatting your diskettes 1-7
Why you should back up your diskettes . . . 1-7

CHAPTER 2. STARTING DOS 2-1
How to Start DOS 2-3
If Your Computer Power is Off 2-3
If Your Computer Power is Already On . . . 2-4
How to Enter the Date 2-5
Specifying the Default Drive 2-6
Automatic Program Execution 2-7
Single-Drive Systems 2-8
Control Keys 2-10
DOS Editing Keys 2-15
Examples of Ways to Use DOS Editing Keys 2-19

CHAPTER 3. DOS COMMANDS 3-1
Introduction 3-3
Types of DOS Commands 3-4
Format Notation 3-5
DOS Command Parameters 3-6
Reserved Device Names 3-8
Global Filename Characters 3-9

Detailed Descriptions of the DOS Commands . . 3-12
Information Common to All DOS Commands 3-12
Batch Processing 3-14
The AUTOEXEC.BAT File 3-15
Creating a .BAT File With Replaceable Parameters 3-16
Executing a .BAT File With Replaceable Parameters 3-17
CHKDSK (Check Disk) Command 3-19
COMP (Compare Files) Command 3-21
COPY Command 3-23
DATE Command 3-27
DIR (Directory) Command 3-29
DISKCOMP (Compare Diskette) Command . . . 3-32
DISKCOPY (Copy Diskette) Command 3-34
ERASE Command 3-37
FORMAT Command 3-38
MODE Command 3-41
PAUSE Command 3-43
REM (Remark) Command 3-44
RENAME Command 3-45
SYS (System) Command 3-46
TIME Command 3-47
TYPE Command 3-49
Summary of DOS Commands 3-50

CHAPTER 4. THE LINE EDITOR (EDLIN) . . . 4-1
Introduction 4-3
How to Start the EDLIN Program 4-4
The EDLIN Command Parameters 4-6
The EDLIN Commands 4-9
Information Common to All EDLIN Commands 4-9
Append Lines Command 4-11
Delete Lines Command 4-12
Edit Line Command 4-16
End Edit Command 4-18
Insert Lines Command 4-19
List Lines Command 4-22
Quit Edit Command 4-26
Replace Text Command 4-27
Search Text Command 4-30
Write Lines Command 4-33
Summary of EDLIN Commands 4-34

CHAPTER 5. THE LINKER (LINK) PROGRAM . 5-1
Introduction 5-3
Files 5-4
Input Files 5-4
Output Files 5-4
VM.TMP (Temporary File) 5-5
Definitions 5-6
Segment 5-6
Group 5-7
Class 5-7
Command Prompts 5-8
Detailed Descriptions of the Command Prompts 5-10
Object Modules: 5-10
Run File: 5-11
List File [run-filename.MAP]: 5-11
Libraries [ ]: 5-12
Publics [No]: 5-13
Line Numbers [No]: 5-13
Stack Size [Object file stack]: 5-13
Load Low [Yes]: 5-14
DSAllocation [No]: 5-15
Special Command Characters 5-16
How to Start LINK 5-17
Before You Begin 5-17
Example of an Automatic Response File . . 5-19
Example Linker Session 5-20
Load Module Memory Map 5-24
How to Determine the Absolute Address of a Segment 5-25
Messages 5-26

CHAPTER 6. THE DEBUG PROGRAM 6-1
Introduction 6-3
How to Start the DEBUG Program 6-4
The DEBUG Command Parameters 6-6
The DEBUG Commands 6-10
Information Common to All DEBUG Commands 6-10
Dump Command 6-12
Enter Command 6-15
Fill Command 6-18
Go Command 6-19
Hexarithmetic Command 6-22
Input Command 6-23
Load Command 6-24

Move Command 6-27
Name Command 6-28
Output Command 6-30
Quit Command 6-31
Register Command 6-32
Search Command 6-37
Trace Command 6-38
Unassemble Command 6-40
Write Command 6-43
Summary of DEBUG Commands 6-47

APPENDIX A. MESSAGES A-3

APPENDIX B. DOS TECHNICAL INFORMATION B-1
DOS Structure B-1
DOS Initialization B-2
The Command Processor B-3
Replacing the Command Processor . . . B-4
Available DOS Functions B-5
Diskette/File Management Notes B-6
The Disk Transfer Area (DTA) B-7
Error Trapping B-7
General Guidelines B-8
Example of Using DOS Functions B-9

APPENDIX C. DOS DISKETTE ALLOCATION . C-1
DOS Diskette Directory C-3
DOS File Allocation Table C-6
How to Use the File Allocation Table C-8

APPENDIX D. DOS INTERRUPTS AND FUNCTION CALLS D-1
Interrupts D-1
Function Calls D-7

APPENDIX E. DOS CONTROL BLOCKS AND WORK AREAS E-1
DOS Memory Map E-1
DOS Program Segment E-2
File Control Block E-7

APPENDIX F. EXE FILE STRUCTURE AND LOADING F-1

INDEX X-1

Preface

This reference manual explains how to use the IBM Personal Computer Disk Operating System (DOS). It provides information on how to issue commands to DOS to create, edit, link, debug, and execute programs.

Prerequisite Publication

Regardless of your background or previous programming experience, you should look at the *Guide to Operations* before reading this manual. The chapter on "Using DOS" should be especially helpful to you before you approach the more detailed information that appears in this manual.

Organization of this manual

This manual has six chapters and six appendixes.

Chapter 1 has some introductory information about DOS and files.

Chapter 2 contains information about starting DOS, as well as directions on how to use the control keys and DOS editing keys.

Chapter 3 contains detailed descriptions of the commands you can issue to DOS. These commands enable you to manage files, maintain diskettes, and create and execute programs.

Chapter 4 describes how to use the Line Editor (EDLIN) program to create, alter, and display source language files and text files.

Chapter 5 describes how to use the linker (LINK) program to link programs together before execution.

Chapter 6 describes how the DEBUG program provides a controlled test environment so you can monitor and control the execution of a program to be debugged; by altering, displaying, and executing object files.

Appendix A lists messages generated by the programs described in this manual.

Appendix B contains general technical information, and introduces the technical information in Appendixes C-E.

Appendix C describes allocation of space on diskettes.

Appendix D describes the system interrupts and function calls.

Appendix E describes control blocks and work areas, including a Memory Map, Program Segment, and File Control Block.

Appendix F contains detailed information about .EXE file structure and loading.

Data Security

The IBM Personal Computer is a powerful and useful tool to help you with your personal and business information processing needs. (As with any information system, inadvertent errors may occur and information may be misused.) We suggest that when processing sensitive or highly valuable information, you take steps to ensure your data and programs are protected from accidental or unauthorized disclosure, modification, destruction or misuse. Simple measures, such as: removing diskettes when not in use, keeping backup copies of valuable information, or installing the equipment in a secure facility, can go a long way to maintain the integrity and privacy of your information.

CHAPTER 1. INTRODUCTION

Contents

What is DOS? 1-3
What are the parts of DOS? 1-3

A Few Words about Files 1-5
What is a file? 1-5
What can I name my files? 1-5
How many files can I have? 1-6
How does DOS keep track of my files? 1-6
Why is this information important to me? . . . 1-6
Formatting your diskettes 1-7
Why you should back up your diskettes 1-7

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, and use the system devices (for example, the printer and the disk drives) attached to your computer.

What are the parts of DOS?

There are four programs on your DOS diskette. These four programs are the "heart" of your DOS:

1. The *boot record.* This program resides at the beginning of your diskette. It is automatically loaded into memory each time you start DOS. The boot record is responsible for loading the rest of DOS. It is placed on all diskettes by the FORMAT program. FORMAT is a program that is supplied with DOS and will be discussed later in this chapter, and also in Chapter 3.

2. The *IBMBIO.COM* program. IBMBIO.COM is an I/O (input/output) device handler program that reads and writes data to and from the computer memory and the devices attached to the computer. This program is on your DOS diskette, but it is not listed when you list the files on the diskette. IBMBIO.COM is also put on your diskette by the FORMAT program and occupies a specific location on the diskette.

3. The *IBMDOS.COM* program. This program also resides on your DOS diskette. Like IBMBIO.COM, its filename does not appear when you list the files in the directory.

IBMDOS.COM 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.

4. The *COMMAND.COM* program. The COMMAND.COM program is a command processor that accepts commands you enter and runs the appropriate programs.

   All the programs on your DOS diskette are designed to run under DOS's control.

A Few Words about Files

What is a file?

A *file* is a collection of related information. A file on your diskette is like a folder in a file cabinet.

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. You might name this file the Employee Master File. A file on your diskette could also contain this information and could also be named the Employee Master File.

All the programs on your diskette reside in files, each with a unique name. You create a file whenever you enter and save data.

Files are kept track of by their names.

What can I name my files?

With few exceptions, you can give your files any names you want. Your diskette *filenames* can be 1-8 characters in length and can be followed by a *filename extension.* Filename extensions start with a period and can be 1-3 characters in length. For example, the Employee Master File could be named EMPMSTR.FLE.

Filenames and filename extensions are discussed in Chapter 3, in the section called "DOS Command Parameters."

How many files can I have?

Each diskette can contain up to 64 files. Files on your diskette vary in size just like files in a file cabinet. If your files contain a lot of information, your diskette fills up with fewer than 64 files.

How does DOS keep track of my files?

The names of your files are kept on your diskette in a system area known as the *directory*. The directory also contains pertinent information concerning the size of your files, their location on the diskette, and the dates they were created or last updated.

The directory occupies four sectors at a specific location on each diskette. For information concerning sectors, refer to the "Using DOS" chapter in your *Guide to Operations*.

Next to the directory is a system area known as a *File Allocation Table*. Its job is to keep track of which sectors belong to which files. The File Allocation Table also keeps track of all available space on the diskette so that you can create new files.

Each diskette has one directory and two copies of the File Allocation Table. If the system has a problem reading the first copy of the File Allocation Table, it reads the second.

Why is this information important to me?

How DOS keeps track of your files is important to you because these system areas are required on all diskettes that DOS is expected to recognize (not just your DOS diskette, but your other diskettes as well). The only way to get this information on a diskette is to use the FORMAT program–it comes on the DOS diskette.

Formatting your diskettes

You must format *every* diskette before it can be used by DOS. You do not need to use FORMAT every time you want to put information on a diskette–only the first time you use a diskette.

FORMAT writes on every sector of your diskette, sets up the directory and File Allocation Table, and puts the boot record program at the beginning of your diskette.

FORMAT also creates a copy of DOS on a new diskette if you specify it in your command. This way, you can create a diskette containing DOS and have plenty of space for your own data on the same diskette. Keep in mind that only DOS files are copied when you run FORMAT–none of the other files you may have on your DOS diskette are copied.

For more information about FORMAT, refer to Chapter 3.

Why you should back up your diskettes

IBM strongly recommends that you make backup copies of all your diskettes. If a diskette somehow becomes damaged, or if files are accidentally erased, you will still have all your information.

> **Note:** Don't forget, if the diskette you plan to use as the *backup* diskette is new, you must FORMAT it before you copy information from the existing diskette.

There are two ways to create a backup diskette:

- Use the DISKCOPY command. DISKCOPY creates an exact image of an entire diskette on another diskette. You can use this command to either copy the DOS diskette or your own diskette. DISKCOPY is the fastest way of copying a diskette because it copies everything, including DOS if it exists, in one operation.

- Use the COPY command to copy all files to a new diskette also. This is a slower method than DISKCOPY, but it produces the same end result with one difference—your files will be written sequentially (one right after the other).

  If you use COPY and you want the new diskette to contain a copy of DOS, you must first FORMAT the diskette with the appropriate option; then use COPY. Unlike DISKCOPY, COPY will not copy the system files for you.

  > **Note:** If either diskette involved in the copy has defective tracks, or if the diskette you want to copy from has had a large amount of file creation/erasure activity, the COPY method is recommended. COPY compensates for the random placement of data caused by the creation/erasure activity and results in better performance.

The dates stored in the directory for each file are unaffected by copying, whether you use COPY or DISKCOPY.

For more information about the COPY and DISKCOPY commands, refer to Chapter 3.

Now that you are more familiar with DOS and files, let's start up DOS.

CHAPTER 2. STARTING DOS

Contents

How to Start DOS 2-3
- If Your Computer Power is Off 2-3
- If Your Computer Power is Already On . . . 2-4
- How to Enter the Date 2-5
- Specifying the Default Drive 2-6

Automatic Program Execution 2-7

Single-Drive Systems 2-8

Control Keys 2-10

DOS Editing Keys 2-15
- Examples of Ways to Use DOS Editing Keys . 2-19

How to Start DOS

There are two ways to start DOS:

- If your computer power is off
- If your computer power is already on

If Your Computer Power is Off

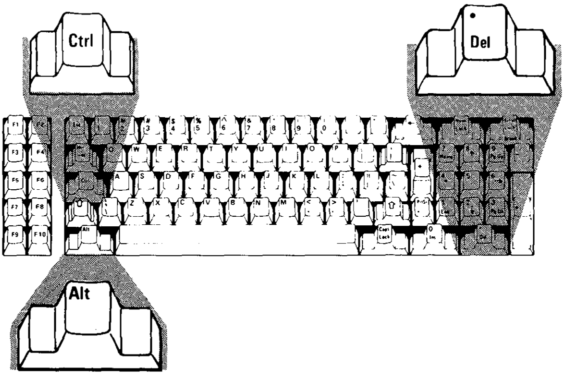
1. Insert your DOS diskette in drive A.
2. Close the drive door.
3. If you have a printer, place the power switch in the **on** position.
4. If your monitor has a separate power switch, place the power switch in the **on** position.
5. Place the system unit power switch in the **on** position.

If Your Computer Power is Already On

1. Insert your DOS diskette in drive A.

2. Close the drive door.

3. Press and hold both the Ctrl and Alt keys; then, press the Del key. Release the three keys. This procedure is known as a *system reset*.



Either of these procedures automatically loads DOS into memory. Loading DOS takes from 3 to 45 seconds, depending on the memory size.

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

Now you must enter the date.

How to Enter the Date

When the command processor is loaded, the following message will be displayed:

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

where:

m is a one- or two-digit number from 1-12
d is a one- or two-digit number from 1-31
y is a two-digit number from 80-99 (the 19 is assumed), or a four-digit number from 1980-2099.

Any date is acceptable as today's date as long as the digits are in the correct ranges and the delimiters (separators) between the numbers are either slashes (/) or hyphens (-).

If you enter an invalid date or delimiter, the system repeats the date prompt.

After you enter a valid date, you see this:

```
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 enter is recorded in the directory entry for any files that you create or change.

A> is the DOS *prompt* from the command processor. Whenever you see A>, the system is waiting for you to enter a command.

You have now completed the steps for starting DOS.

> **Note:** If you did not receive the system messages described, repeat the steps for starting DOS.

Specifying the Default Drive

The A in the prompt designates the *default drive.* DOS searches the diskette located in the default drive to find any filenames that you enter unless you specify another drive.

You can change the default drive in the prompt by entering the new designation letter followed by a colon. For example:

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

Now, B is the default drive. DOS searches the diskette located in drive B to find any filenames that you enter, unless you specify a drive.

Remember, if you do not specify a drive when you enter a filename, the system automatically searches the diskette located in the default drive.

Automatic Program Execution

You may want to start a specific program every time you start DOS. You can do this with the DOS command processor by using *automatic program execution.*

Every time you start up DOS, 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* that is automatically executed whenever you start the system. With this facility, you can execute programs or commands immediately every time you start DOS.

If DOS finds the AUTOEXEC.BAT file, the file is immediately executed by the command processor. The date prompt will be bypassed.

If DOS does not find the AUTOEXEC.BAT file, it issues the date prompt. Refer to "Batch Processing" in Chapter 3 for details on how to create an AUTOEXEC.BAT file.

Single-Drive Systems

On a single-drive system, you enter the commands the same way you would on a multi-drive system.

You should think of the single-drive system as having *two* drives (drive A and drive B). Instead of A and B representing two physical drives as on a multi-drive system, the A and B represent diskettes.

If you specify drive B when the "drive A diskette" was last used, you are prompted to insert the diskette for drive B. 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, the system prompts you to insert the "drive A diskette."

The same procedure is used if a command is executed from a batch file. The system waits for you to insert the appropriate diskette and press any key before it continues.

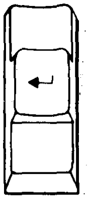
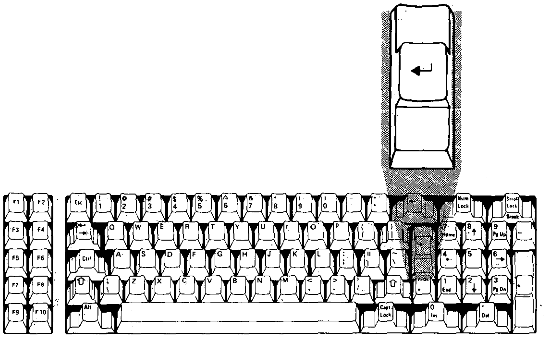
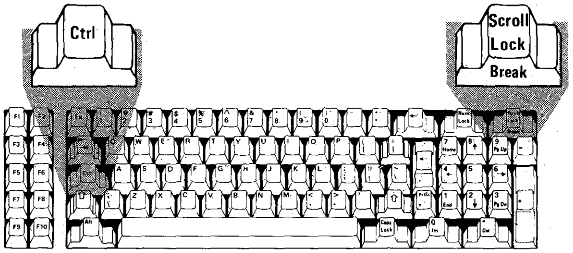
> **Note**: Remember that the letter displayed in the system prompt represents the default drive where DOS looks to find a file whose name is entered without a drive specifier. The letter in the system prompt does *not* represent the last diskette used.
>
> For example, assume that A: is the default drive. If the last operation performed was DIR B:, DOS believes the "drive B diskette" is still in the drive; however, the system prompt is still A> because A is still the default drive. If you issue DIR A:, DOS prompts you for the "drive A diskette."

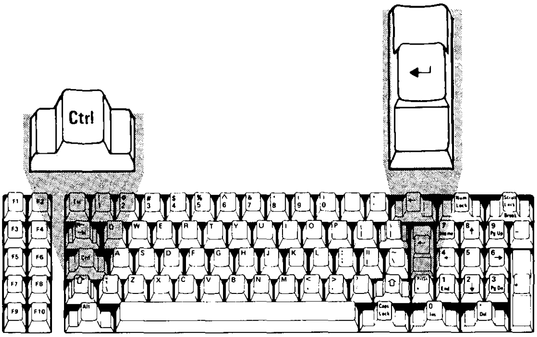
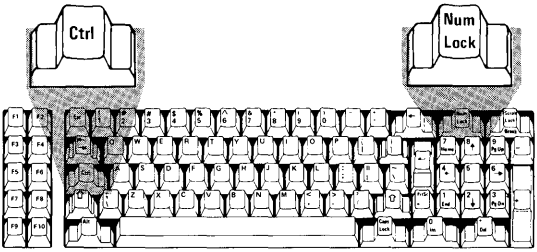
Now that you know how to start the system and specify drives, you should learn about the keys on your keyboard that you can use with DOS.

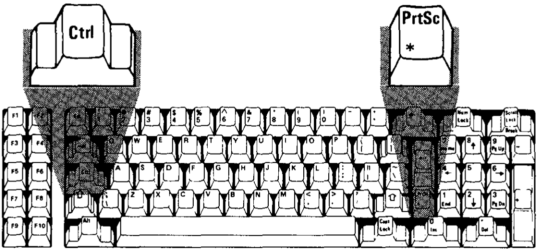
Control Keys

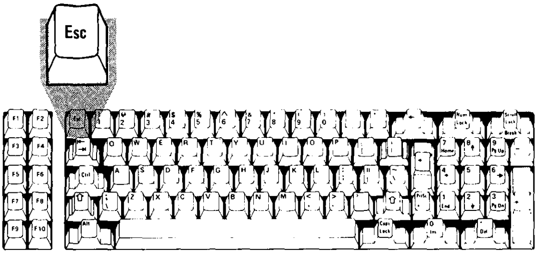
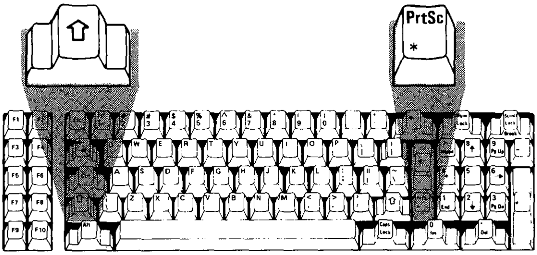
Use the control keys when you are entering commands or input lines to any program. Where two keys are specified, for example Ctrl-Break, you must press and hold down the first key and then press the second key.

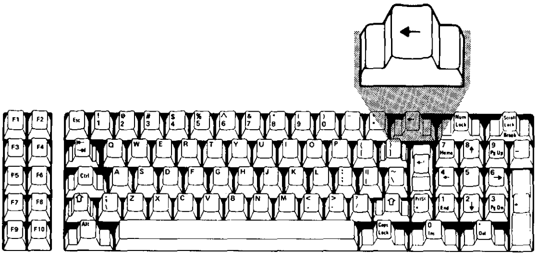
Here is a summary of the control keys, their functions, and their location on the keyboard:

Control Key	Function
	This is the Enter key. Once you press the Enter key, the displayed line is sent to the requesting program. 
Ctrl-Break	Ends the current operation. 

Control Key	Function
Ctrl-Enter	Allows you to go to the next display line on the screen to continue entering the line being typed. 
Ctrl-NumLock	Suspends system operation. You must press any character key to resume operation. This is useful when a large amount of screen output is being generated. You can press Ctrl-NumLock to temporarily suspend the display of your output so you can review it. You can then press any other character key to restart the display. 

Control Key	Function
Ctrl-PrtSc	These keys serve as an on/off switch for sending display output to the printer as well as to the screen. You can press these keys to print display output on the printer and press them again to *stop* printing display output on 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 you run Disk and Advanced BASIC. 

Control Key	Function
Esc	Cancels the current line and moves to the next display line. A back slash (\\) is displayed to indicate the cancelled line. 
Shift-PrtSc	Sends a copy of what is currently displayed on the screen to the printer. This, in effect, prints a “snapshot” of the screen. 

Control Key	Function
←	Backspaces and removes a character from the screen. This is the key to the left of NumLock, *not* key 4 on the numeric keypad. 

DOS Editing Keys

Use the DOS editing keys to make corrections to commands and input lines as they are being entered.

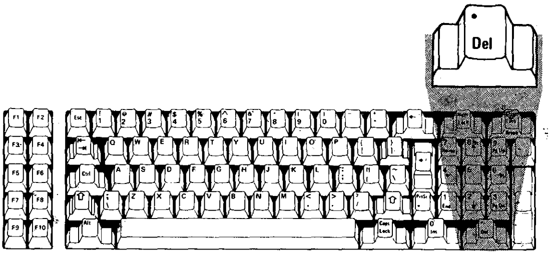
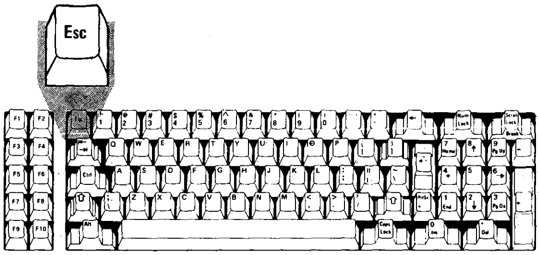
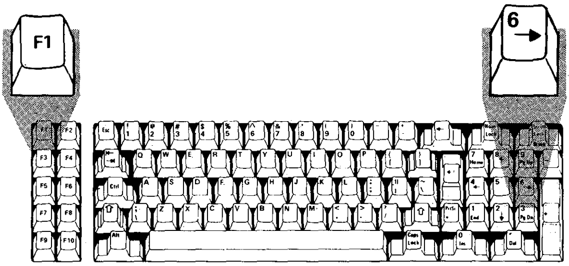
The DOS editing keys are used to edit *within* a line. The Line Editor (EDLIN) program operates on *complete lines* within a file or document. When you are working with EDLIN and want to edit within a line; however, use the DOS editing keys. For more information about EDLIN, refer to Chapter 4.

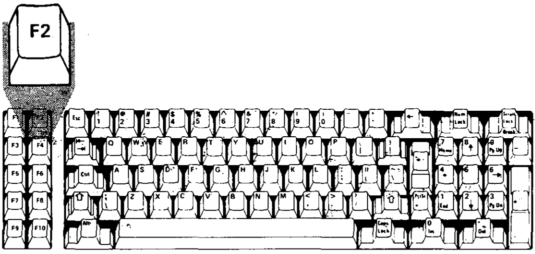
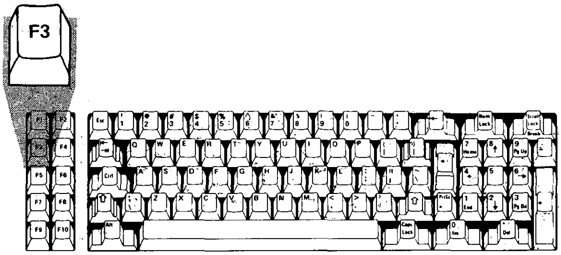
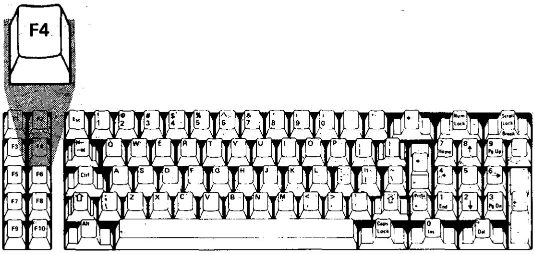
> **Note:** Some word processing programs define special editing rules; therefore, the DOS editing keys may not work as described in this chapter. You can also define special editing rules when using the BASIC Program Editor used while programming in BASIC.

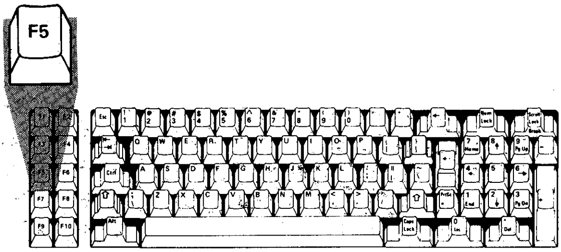
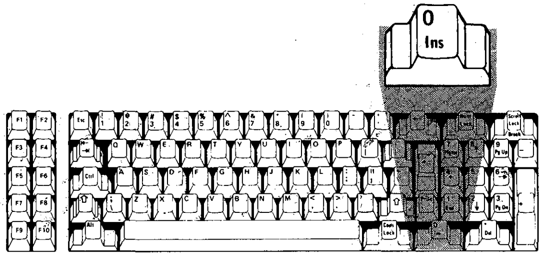
Any line you enter from the keyboard is retained in an input buffer when you press Enter. The line is then made available to your program for processing.

Since the line remains in the input buffer, you can use that line as a *template* for editing purposes. The DOS editing keys operate on that copy of the line. You can repeat or change the line by using the DOS editing keys, or you can enter an entirely new line.

Here is a summary of the DOS editing keys, their function, and their location on the keyboard:

DOS Editing Key	Function
Del	Skips over one character in the template. The cursor does not move. 
Esc	Cancels the line currently being displayed. The template remains unchanged. 
F1 or →	Copies one character from the template and displays it. 

DOS Editing Key	Function
F2	Copies all characters up to a specified character. 
F3	Copies all remaining characters from the template to the screen. 
F4	Skips over all characters up to a specified character. (F4 is the opposite of F2.) 

DOS Editing Key	Function
F5	Accepts an edited line for continued editing—the currently displayed line becomes the template, but it is not sent to the requesting program. 
Ins	Allows you to insert characters within a line. 

Examples of Ways to Use DOS Editing Keys

The following examples show how you use the DOS editing keys with the Line Editor (EDLIN) program.

If you want to try these examples, you must use the EDLIN program. The EDLIN program is on your DOS diskette and is discussed in Chapter 4. You do not have to review the EDLIN chapter to complete these examples–just follow the steps provided.

> **Note:** Because the DOS diskette shipped with your IBM Personal Computer is *write protected*, you cannot create the file used in the following examples on that diskette. You must use a *copy* of your DOS diskette to complete these examples. Refer to the section called "Write Protect Notch" in the DOS section of your *Guide to Operations* for more information about write protected diskettes.

To Start EDLIN:

1. Insert your DOS diskette into drive A.

2. Create a file named EXAMPLES.

If you want the EXAMPLES file to reside on the diskette in your default drive, enter:

```
EDLIN EXAMPLES
```

or

If you want the EXAMPLES file to reside on the diskette in another drive, you must specify the drive, as in:

```
EDLIN B:EXAMPLES
```

This command tells DOS to load the EDLIN program and create a file called EXAMPLES.

The following message and prompt will be displayed:

```
New file
*_
```

Notice that the prompt for EDLIN is an asterisk (*).

3. Now, enter the letter **I**.

   This tells EDLIN that you want to begin *inserting* lines in the file named EXAMPLES.

   The screen looks like this:

```
New file
*I
   1:*_
```

4. Type **This is a mailorder file.** on line 1 and press Enter.

5. Type **Editing is easy.** on line 2 and press Enter.

   You now have two lines of text in your EXAMPLES file.

6. Press the Ctrl-Break keys.

   Pressing Ctrl-Break will end the insert mode of operation and return you to the EDLIN prompt.

7. Enter the number **1**.

   This tells EDLIN that you want to display line 1 on the screen.

   The screen looks like this:

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

You are now ready to begin the examples.

> **Note:** If you encounter any problems while trying these examples, press the Ctrl-Break keys. The EDLIN prompt will be displayed and you can start over.

Example 1

Let's delete the first two characters in the word **This** and then copy the remainder of the line.

1. Press the Del key twice to delete the first two characters.

2. Press F3 to copy the remainder of the line to the screen. The screen looks like this:

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

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt. (The changes you made to line 1 will not be saved.)

2. Enter the number 1.

Example 2

Now we'll change line 1; then, using Esc, we will cancel the change. A back slash (\) will be displayed to indicate that the displayed line has been cancelled.

> **Note:** If the insert mode is on, the system automatically turns it off when you use Esc.

The screen looks like this:

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

To change line 1 to **Sample file**:

1. Type **Sample file**, but do *not* press Enter.

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

2. To cancel the line we just entered, press the Esc key.

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

   Now we can continue to edit the original line **This is a mailorder file**.

3. Press F3 to copy the original line to the screen.

The screen looks like this:

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

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number 2.

Example 3

Now let's copy one character by using F1 or →. (F1 or → is the opposite of Del. Del skips over one character in the template.)

The screen looks like this:

```
2:*Editing is easy.
2:*_
```

1. Press the F1 or → key three times.

   The screen looks like this:

   ```
   2:*Editing is easy.
   2:*Edi_
   ```

   Each time you press the F1 or → key, one more character appears.

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number 2.

Example 4

Now let's use the F2 key. Remember, the F2 key copies all characters from the template to the screen up to, but not including, the first occurrence of a specified character.

You must always specify a character when using this key. If the specified character is not present in the template, nothing is copied.

The screen looks like this:

```
2:*Editing is easy.
2:*_
```

1. Press the F2 key and enter the letter **g**.

   The screen looks like this:

   ```
   2:*Editing is easy.
   2:*Editin_
   ```

   Now we'll copy all the remaining characters in the template to the screen by using the F3 key.

   (If you pressed Enter now, only **Editin** would be saved in the EXAMPLES file as line 2.)

2. Press the F3 key.

   The screen looks like this:

   ```
   2:*Editing is easy.
   2:*Editing is easy._
   ```

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number **1**.

Example 5

Now let's scan and locate specific characters within the template by using the F4 key. This is a way to skip over characters. The cursor does not move when you use this key and no characters are displayed.

You must always specify a character after you press the F4 key. If the specified character is not present in the template, no characters in the template will be skipped.

We will also use the F3 key to copy the remaining characters in the template to the screen.

The screen looks like this:

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

1. Press the F4 key and enter the letter o. (No characters are displayed.)

2. Press the F3 key to copy the remainder of the line.

   The screen looks like this:

   ```
   1:*This is a mailorder file.
   1:*order file._
   ```

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number 1.

Example 6

Now we'll move the currently displayed line into the template by using the F5 key. Pressing F5 is the same as pressing Enter, except that the line is *not* sent to your program. An @ character is displayed to indicate that the new line is now the template.

> **Note:** If the insert mode is on, the system automatically turns it off when you use F5.

Once you press F5, you can continue to make changes to a line. When you are finished, press Enter to send the line to your program.

The screen looks like this:

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

1. Type **This is not a sample file.**

   The screen looks like this:

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

2. Press F5.

   The result is:

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

   The replacement line **This is not a sample file.** is now in the template. The replacement line is acceptable, but let's continue to edit it.

3. To remove the word **not** from the replacement line, press F1 eight times:

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

4. Press Del four times to remove one blank space and the word **not**.

5. Press F3 to copy the remaining characters to the screen.

   The screen looks like this:

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

6. Press Enter to make the replacement line **This is a sample file.** the template in place of the original line *and* to send the line to your program.

   (If you want to do more editing without sending the line to your program, press F5 again to put the displayed line into the template.)

   Note: Pressing Enter *immediately* after pressing F5 empties the template.

If you want to continue with the next example:

1. Press Ctrl-Break to return to the EDLIN prompt.

2. Enter the number **1**.

Example 7

Let's look at an example using the Ins key. The Ins key serves as an on/off switch for entering and leaving insert mode. You can press the Ins key to enter insert mode, and press the Ins key again to leave the insert mode.

While in the insert mode of operation, any characters that you enter are *inserted* in the line being displayed. The characters do not *replace* characters in the template.

When you are not in the insert mode of operation, any characters that you enter *replace* characters in the template. If you are entering characters at the end of a line, the characters will be added to the line.

The screen looks like this:

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

Let's change the word **sample** to **salary**.

1. Press the F2 key and enter the letter **m**.

   The screen looks like this:

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

2. Press the Ins key and enter the characters **lary**.

   The screen looks like this:

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

   Notice that the characters **lary** were inserted, but no characters from the template were replaced.

3. Now, press Ins again to leave the insert mode.

4. Enter one blank space and the three characters **tax**.

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

5. Press F3 to copy the remaining characters in the template to the screen.

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

   Notice that we *inserted* **lary** and we *replaced* **mple** with **tax**.

6. Now press Enter to make the replacement line the template in place of the original line and send the line to the requesting program.

You have now completed the examples.

To return to the A> prompt:

1. Press Ctrl-Break.

2. Enter the letter **Q**.

   Q tells DOS that you don't want to save the EXAMPLES file and that you want to *quit* the editing session. EDLIN will prompt you with this message:

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

   to make sure you don't want to save the file.

3. Enter the letter **N**.

The next chapter contains detailed information about the DOS commands.

CHAPTER 3. DOS COMMANDS

Contents

Introduction 3-3

Types of DOS Commands 3-4

Format Notation 3-5

DOS Command Parameters 3-6
Reserved Device Names 3-8
Global Filename Characters 3-9
The ? Character 3-9
The * Character 3-9
Examples of Ways to Use ? and * . . . 3-10

Detailed Descriptions of the DOS Commands . . 3-12
Information Common to All DOS Commands 3-12

Batch Processing 3-14
The AUTOEXEC.BAT File 3-15
Creating a .BAT File With Replaceable Parameters 3-16
Executing a .BAT File With Replaceable Parameters 3-17

CHKDSK (Check Disk) Command 3-19

COMP (Compare Files) Command 3-21

COPY Command 3-23

DATE Command 3-27

DIR (Directory) Command 3-29

DISKCOMP (Compare Diskette) Command . . . 3-32

DISKCOPY (Copy Diskette) Command 3-34

ERASE Command 3-37

FORMAT Command 3-38

MODE Command 3-41

PAUSE Command 3-43

REM (Remark) Command 3-44

RENAME Command 3-45

SYS (System) Command 3-46

TIME Command 3-47

TYPE Command 3-49

Summary of DOS Commands 3-50

Introduction

This chapter explains how to use the DOS commands.

You can use DOS commands to:

- Compare, copy, display, erase, and rename files.
- Compare, copy, and format diskettes.
- Execute 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 the system to wait.

Types of DOS Commands

There are two types of DOS commands:

- Internal
- External

Internal commands execute immediately because they are built-in to the command processor.

External commands reside on diskette as program files; therefore, they must be read from diskette before they execute. This means that the diskette containing the command must already be in a drive, or DOS is unable to find the command.

Any file with a filename extension of .COM or .EXE is considered an external command. By adding the extension .COM or .EXE to your filenames, you can develop and add your own unique commands to the system.

When you enter an external command, do not include the filename extension.

Format Notation

We will use the following notation to indicate how the DOS commands should be formatted:

- You must enter any words shown in capital letters. These words are called *keywords* and must be entered exactly as shown. You can, however, enter keywords in any combination of upper/ lowercase. DOS will automatically convert keywords to uppercase.

- You must supply any items shown in lowercase italic letters. For example, you should enter the name of *your* file when *filename* is shown in the format.

- Items in square brackets ([ ]) are optional. If you want to include optional information, you do not need to type the brackets, only the information inside the brackets.

- An ellipsis (. . .) indicates that you can repeat an item as many times as you want.

- You must include all punctuation (except square brackets) such as commas, equal signs, question marks, colons, or slashes where shown.

DOS Command Parameters

Parameters are items that can be included in your DOS command statements. They are used to specify additional information to the system. Some parameters are required in your commands, others are optional. If you do not include some parameters, DOS provides a default value. Default values that DOS provides are discussed in the detailed descriptions of the DOS commands. Use the following parameters in your DOS command statements:

Parameter	**Definition**
d:	Denotes when you should specify a drive. Enter a drive letter followed by a colon to specify the drive. For example, A: represents the first drive on your system, B: represents the second. If you omit this parameter, DOS assumes the *default* drive.
filename	Diskette filenames are 1-8 characters in length, and can be followed by a filename extension. The following characters can be used for filenames: A-Z 0-9 $ & # @ ! % ' ( ) - < > { } _ \ ^ ~ ¦ ‘ Any other characters are invalid. An invalid character is assumed to be a delimiter, in which case the filename is truncated. Refer also to "Reserved Device Names" in this chapter for more information about filenames.

Parameter	Definition
.ext	The optional filename extension consists of a period and 1-3 characters. When used, filename extensions immediately follow filenames. The following characters can be used for filename extensions: A-Z 0-9 $ & # @ ! % ’ ( ) - < > { } _ \ ^ ~ ¦ ‘ Any other characters are invalid. Remember to include the extension when you refer to a file that has a filename extension; otherwise, DOS will be unable to locate the file.
filespec	[d:]filename[.ext] Examples: **B:myprog.COB** **A:yourprog** **DATAFILE.pas** **cobfile**

Reserved Device Names

Certain names have special meaning to DOS. DOS reserves the following names as system devices:

Reserved Name	Device
CON:	Console keyboard/screen. If used as an input device, you can press the F6 key; then press the Enter key to generate an end-of-file indication, which ends CON: as an input device.
AUX: or COM1:	First Asynchronous Communications Adapter port.
LPT1: or PRN:	Printer (as an output device only).
NUL:	Nonexistent (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.

Notes:

1. The reserved device names can be used in place of a filename.

2. Any drive specifier or filename extension entered with these device names will be ignored.

3. The colon following the reserved device word is optional.

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 in a filename extension indicates that any character can occupy that position. For example,

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

lists all directory entries on the default drive with filenames that have five characters, begin with AB, have any next character, are followed by DE, and have an extension of XYZ.

Here are some examples of the files that might be listed by the DIR command:

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

The * Character

An * in a filename or in a filename extension indicates 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 filenames may be from 2-8 characters in length.

Here are some example files that might be listed by the DIR command:

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

Examples of Ways to Use ? and *

Example 1

To list the directory entries for all files named INPUT on drive A (regardless of their filename extension), enter:

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

Example 2

To list the directory entries for all files on drive A (regardless of their filenames) with a filename extension of XYZ, enter:

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

Example 3

To list the directory entries for all files on drive A with filenames beginning with ABC and extensions beginning with E, enter:

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

Detailed Descriptions of the DOS Commands

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

Information Common to All DOS Commands

The following information applies to all DOS commands:

- With the exception of the DATE and TIME commands, commands are usually followed by one or more parameters.

- Commands and parameters may be entered in uppercase or lowercase, or a combination of both.

- Commands and parameters *must* be separated by delimiters (space, comma, semicolon, equal sign, or the tab key). The delimiters can be different within one command. For example, you could enter:

  ```
  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.

- In this book, we will usually use a space as the delimiter in the commands for readability.

- Files are not required to have filename extensions when you create or rename them; however, you must include the filename extension when referring to a file that *has* a filename extension.

- You can end commands while they are running by pressing Ctrl-Break. 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 you press the Enter key.

- Global filename characters and device names are not allowed in a command name. You may only use them in command parameters.

- For commands producing a large amount of output, you can press Ctrl-NumLock to suspend the display of the output. You can then press any other key to restart the display.

- You can use the control keys and the DOS editing keys described in Chapter 1 while entering the DOS commands.

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

- Drives will be referred to as *source* drives and *target* drives. A source drive is the drive you will be transferring information *from*. A target drive is the drive you will be transferring information *to*.

Batch Processing

Purpose: Executes the commands contained in the specified file from the designated or default drive.

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

Type: Internal External
***

Remarks: A batch file is a file containing one or more commands that DOS executes one at a time. All batch files must have a filename extension of .BAT.

You can pass parameters to the filename.BAT file when the file executes. Therefore, the file can do similar work with different data during each execution.

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

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 enter an extension.

3. The commands in the file named filename.BAT are executed.

4. If you press Ctrl-Break while in batch mode, this prompt appears:

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

   If you press Y, the remainder of the commands in the batch file are ignored and the system prompt appears.

   If you press N, only the current command ends and batch processing continues with the next command in the file.

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

The AUTOEXEC.BAT File

The AUTOEXEC.BAT file is a special batch file. When you start or restart DOS, the command processor searches for the AUTOEXEC.BAT file. If this file is present on the DOS diskette, DOS automatically executes the file whenever you start DOS.

For example, if you want to load BASIC and run a program called MENU automatically, create an AUTOEXEC.BAT file as follows:

1. Enter:

   ```
   COPY CON: AUTOEXEC.BAT
   ```

   This statement tells DOS to copy the information from the console (keyboard) into the AUTOEXEC.BAT file.

2. Now, enter:

   ```
   BASIC MENU
   ```

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

3. Press the F6 key; then press the Enter key to put the command BASIC MENU in the AUTOEXEC.BAT file.

   The MENU program will now run 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. Remember, you can enter any DOS command, or series of commands, in the AUTOEXEC.BAT file.

> **Note:** If you use AUTOEXEC, DOS does not prompt you for the current date unless you include a DATE command in the AUTOEXEC.BAT file.

Creating a .BAT File With Replaceable Parameters

Within a batch file you may include *dummy* parameters that can be replaced by values supplied when the batch file executes.

For example, enter:

```
A>Copy con: ASMFILE.BAT
Copy %1.MAC %2.MAC
Type %2.PRN
Type %0.BAT
```

Now, press F6; then press Enter.

The system responds with this message:

```
1 File(s) copied
A>_
```

The file ASMFILE.BAT, which consists of three commands, now resides on the diskette in the default drive.

The dummy parameters %0, %1, and %2 are replaced sequentially by the parameters you supply when you execute the file. The dummy parameter %0 is always replaced by the drive designator, if specified, and the filename of the batch file.

Notes:

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

2. 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.

Executing a .BAT File With Replaceable Parameters

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

For example, you can enter:

```
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 entered each of the three commands (in the ASMFILE.BAT file) from the console with their parameters, as follows:

```
Copy A:PROG1.MAC 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 the filename of the batch file.

CHKDSK (Check Disk) Command

Purpose: Analyzes the directory and the File Allocation Table on the designated or default drive and produces a diskette and memory status report.

Format: CHKDSK [*d*:]

Type:

Internal	External
	***

Remarks: CHKDSK temporarily makes the drive specified in *d*: the default drive. If CHKDSK ends prematurely, the default drive changes to the drive that CHKDSK was checking. CHKDSK might end prematurely if you replied A to a diskette error message.

After checking the diskette, any error messages are displayed. A complete listing of error messages can be found in Appendix A.

The following status report is also displayed:

```
      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.

The system does not wait for you to insert a diskette. CHKDSK assumes that the diskette to be checked is in the specified drive. Therefore, on a single-drive system, it is especially important that the specified drive is different from the default drive, unless you are checking the DOS diskette itself.

You should run CHKDSK occasionally for each diskette to ensure the integrity of the file structures.

COMP (Compare Files) Command

Purpose: Compares the contents of one file to the contents of another file.

> **Note:** This command compares two *files*; the DISKCOMP command compares two *entire diskettes.*

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

Type:

Internal	External
	***

Remarks: The files that you compare may be on the same drive or on different drives. Both filenames are optional, but if you omit them, DOS prompts you for them. You are also prompted to insert the appropriate diskettes. COMP waits for you to press a key before it starts comparing the files.

COMP compares the files byte-for-byte. Unequal bytes result in error messages that give the hexadecimal offset of the unequal comparison and the bytes that were compared, as follows:

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

In this example, File 1 is the first filename entered; File 2 is the second.

After ten unequal comparisons, COMP concludes that further comparing would be useless; processing ends; and the following message is displayed:

```
10 Mismatches - aborting compare
```

After a successful comparison, COMP displays:

```
Files compare ok
```

After a comparison ends, COMP displays:

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

You now have the option to compare two more files or to end the comparison. If you want to compare two more files, enter the filespec of the first file. COMP prompts you for the second.

If you want to end COMP processing, press Enter.

Notes:

1. The two files you want to compare can have the same name—provided they are on different diskettes. In this case the specified drives must be different.

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

3. Use of the global characters ? and * in the filenames do not cause multiple file comparisons. Only the first file matching each name is compared. For more information on the global characters, refer to "Global Filename Characters" in this chapter.

4. A comparison does not take place if the file sizes are different. DOS prompts you to compare other files.

COPY Command

Purpose: Copies one or more files to another diskette and optionally, gives the copy a different name if you specify it in the COPY command.

COPY also copies files to the same diskette. In this case, you *must* give the copies different names; otherwise, the COPY is not permitted.

You can also use COPY to transfer data between any of the system devices. (An example of how to copy information that you enter at the keyboard to a diskette file is provided at the end of the description of COPY.)

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

Type: Internal External
***

Remarks: The first parameter, *filespec*, is the source file. The second parameter, [*d*:] [*filename*[.*ext*]], is the target file.

You can use the global characters ? and * in the filename and in the extension parameters of both the original and duplicate files. For more information about global characters, refer to "Global Filename Characters" in this chapter.

The COPY command has two format options:

Option 1

Use this option to copy a file with the copied file having the *same* filename and extension. For example:

COPY *filespec*

or

COPY *filespec d*:

In the first example, we want to copy a file to the default drive. In the second example, we specified the target drive. In both examples, because we did not specify the second filename, the copied file will have the same filename as the source file. Because we did not specify a name for the second file, the source drive and the target drive must be different; otherwise, the copy is not permitted.

For example, assume the default drive is A:. The command:

```
COPY B:MYPROG
```

copies the file MYPROG from the diskette in drive B, to the diskette in default drive A, with no change in the filename. The command:

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

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

Option 2

Use this option when you want the copied file to have a different name from the file that is being copied. For example:

COPY *filespec filename*[*.ext*]

or

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

In the first example, we copied a file, *filespec*, and renamed the copy, *filename*[*.ext*]. We did not specify a drive, so the default drive was used. In the second example, we copied a file and renamed the copy also. In this example, we did specify the target drive. Because we changed the name of the file, the source drive and the target drive do not have to be different.

For example:

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

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

You can also use reserved device names for the copy operation. For example:

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

Also, NUL: can be used in any variation.

Refer to "Reserved Device Names" for information about system devices.

Example: This example shows how to use COPY to put what you enter at the console into a diskette file:

```
A>COPY CON: fileA
Type a line and press Enter.
Type your next line and press Enter.
●
●
●
Type your last line and press Enter.
Now, press F6 and then press Enter.
```

When you press F6, and then press Enter, the COPY operation ends and saves the information you entered. In this example, the information is saved in a file named fileA.

DATE Command

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

Format: DATE

Type:

Internal	External
	***

Remarks: The DATE command issues the following prompt:

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

where:

m is a one- or two-digit number from 1-12
d is a one- or two-digit number from 1-31
y is a two-digit number from 80-99 (the 19 is assumed)
or a four-digit number from 1980-2099

You can change the date from the console or from a batch file. Remember, when you start the system, it does not prompt you for the date if you use an AUTOEXEC.BAT file. You may want to include a DATE command in that file. For more information about the AUTOEXEC.BAT file, refer to "Batch Processing" in this chapter.

Notes:

1. To leave the date as is, press Enter.

2. The valid delimiters within the date are hyphens (-) and slashes (/).

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

4. If you enter an invalid date or delimiter, you receive an **Invalid date** message.

Example: In this example, once you press Enter, the date known to the system is 7/24/82.

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

DIR (Directory) Command

Purpose: Either lists all the directory entries, or only lists those for specified files. The information provided in the display for each file includes its size in decimal bytes and the date the file was last written to.

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

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

Type:

Internal	External
***	

Remarks: You can use the global characters ? and * in the filename and extension parameters. For more information about the global characters, refer to "Global Filename Characters" in this chapter.

The DIR command has two format options:

Option 1

Use this option to list all the files in a directory. For example:

```
DIR

  or

DIR d:
```

In the first example, we want to list all directory entries on the default drive. In the second example, we want to list all directory entries on the specified drive.

The directory listing might look like this:

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

Option 2

Use this option to list selected files from a directory. For example:

DIR *filename.ext*

or

DIR *d:filename.ext*

In the first example, we want to list all the files in the directory of the default drive that have the specified filename.ext. In the second example, we want to list all the files in the directory of the specified drive that have the specified filename.ext.

Using the previous example, if you enter:

```
DIR FILE3.A
```

the screen might look like this:

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

If you enter:

```
DIR *.A
```

the screen might look like this:

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

DISKCOMP (Compare Diskette) Command

Purpose: Compares the contents of the diskette in the first drive to the contents of the diskette in the second drive. Usually, you would run DISKCOMP after a DISKCOPY operation to ensure that the two diskettes are identical.

> **Note:** This command compares two *entire diskettes*; the COMP command compares two *files*.

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

Type:

Internal	External
	***

Remarks: You can specify the same drive or different drives in this command. If you specify the same drives, a single-drive comparison is performed. You are prompted to insert the diskettes at the appropriate time. DISKCOMP waits for you to press any key before it continues.

DISKCOMP compares all 40 tracks on a track-for-track basis and issues a message if the tracks are not equal. The message indicates the unequal track number (0-39).

After completing the comparison, DISKCOMP prompts:

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

If you press Y, the next comparison is done on the same drives that you originally specified, after you receive prompts to insert the proper diskettes.

To end the command, press N.

Notes:

1. If you omit both parameters, a single-drive comparison is performed on the default drive.

2. If you omit the second parameter, the default drive is used as the secondary drive. If you specify the default drive in the first parameter, this also results in a single-drive comparison.

3. On a single-drive system, all prompts are for drive A, regardless of any drive specifiers entered.

4. 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 diskette 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 inserted.

5. DISKCOMP does not usually issue a **Diskettes compare OK** message if you try to compare a backup diskette created by the COPY command with the diskette you copied from. The COPY operation produces a copy that contains the same information, but places the information at different locations on the target diskette from those locations used on the source diskette. In this case, you should use the COMP command to compare individual files on the diskettes.

DISKCOPY (Copy Diskette) Command

Purpose: Copies the contents of the diskette in the source drive to the diskette in the target drive.

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

Type: Internal External
***

Remarks: The first parameter you specify is the source drive. The second parameter is the target drive.

You can specify the same drives or you may specify different drives. If the drives are the same, a single-drive copy operation is performed. 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)_
```

If you press Y, the next copy is done on the same drives that you originally specified, after you are prompted to insert the proper diskettes.

To end the command, press N.

Notes:

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

2. If you omit the second parameter, the default drive is used as the target drive.

3. If you omit the second parameter and you specify the default drive as the source drive, a single-drive copy operation is performed.

4. On a single-drive system, all prompts will be for drive A, regardless of any drive specifiers you may enter.

5. 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.

6. Diskettes that have had a lot of file creation and deletion activity become fragmented because diskette space is not allocated sequentially. The first free sector found is the next sector allocated, regardless of its location on the diskette.

   A fragmented diskette 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 to the diskette in drive B.

7. You should run DISKCOMP after a successful DISKCOPY to ensure that the diskettes are identical.

ERASE Command

Purpose: Deletes the file with the specified filename from the designated drive, or deletes the file from the default drive if no drive was specified.

Format: ERASE *filespec*

Type: Internal External
***

Remarks: You can use the global characters ? and * in the filename and in the extension. Global characters should be used with caution, however, because multiple files can be erased with a single command. For more information about global characters, refer to "Global Filename Characters" in this chapter.

To erase all files on a diskette, enter:

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

> **Note:** The system files IBMBIO.COM, IBMDOS.COM, and BADTRACK cannot be erased.

Example: In this example, the file **myprog.1** will be erased from the diskette in drive A.

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

FORMAT Command

Purpose: 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.

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

Type:

Internal	External
	***

Remarks: You must run FORMAT on all new diskettes before they can be used by DOS.

If you specify /S in the FORMAT command, the operating system files are also copied from the diskette in the default drive to the new diskette in the following order:

IBMBIO.COM
IBMDOS.COM
COMMAND.COM

Notes:

1. Formatting 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 the tracks 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, you can analyze the status report displayed by the CHKDSK command.

   CHKDSK should report zero disk files if you did not specify /S in the FORMAT command. If one disk file is reported, that means defective tracks were found and the file being reported is BADTRACK.

   If you did specify /S in the FORMAT command, CHKDSK should report three files present; a report of four files means defective tracks were found.

Example: By issuing the following command, the diskette in drive B will be formatted and the operating system files will also be copied:

```
A>FORMAT B:/S
```

The system issues the following message:

```
Insert new diskette for drive B:
and strike any key when ready
```

After you insert the appropriate diskette and strike any key, the system issues this message:

```
Formatting...
```

while the diskette formatting is taking place.

Once the formatting is complete, the system issues this message:

```
Formatting. . .Format complete
System transferred
Format another (Y/N)?_
```

Press Y to format another diskette.

Press N to end the FORMAT program.

MODE Command

Purpose: Sets the mode of operation on a printer or on a display connected to the Color/Graphics Monitor Adapter.

> **Note:** This command has no effect on a display connected to the IBM Monochrome Display Printer Adapter.

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

Type:

Internal	External
	***

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

The MODE command has two format options:

Option 1 (For the Printer)

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

where:

# is 1, 2, or 3 (the 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.

The power-on default options for the printer 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 used to align the display.

For readability, you can shift the display one character (for 40-column) or two characters (for 80-column) in either direction. If you specify T in the MODE command, a prompt will ask you 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

Purpose: Suspends system processing and issues the message **Strike a key when ready. . . .**

Format: PAUSE [*remark*]

Type:

Internal	External
***	

Remarks: You can insert PAUSE commands within a batch file to display messages and to give you the opportunity to change diskettes between commands. To resume execution of the batch file, press any key *except* Ctrl-Break. (Ctrl-Break ends processing.)

If you include the optional remark, the remark is also displayed. The optional remark can be any string of characters up to 121 bytes long.

You can control how much of a batch file you want to execute by placing PAUSE commands at strategic points in the file. At each PAUSE command, the system stops and gives you time to decide whether to end processing. To end processing, press Ctrl-Break. To continue processing, press any other key.

Example: If you enter this PAUSE command in a batch file, the following message is displayed:

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

This PAUSE enables you to change diskettes between commands.

REM (Remark) Command

Purpose: Displays remarks from within a batch file.

Format: REM [*remark*]

Type:

Internal	External
***	

Remarks: The remarks are displayed when the batch execution reaches the remark.

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

You can use REM commands without remarks for spacing within your batch file, for readability.

Example: If the following REM command is issued in a batch file, this remark is displayed:

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

RENAME Command

Purpose: Changes the name of the file specified in the first parameter to the name and extension given in the second parameter. If a valid drive is specified in the second parameter, the drive is ignored.

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

Type:

Internal	External
***	

Remarks: You can use the global characters ? and * in the parameters. For more information about global characters, refer to "Global Filename Characters" in this chapter.

Example: The command:

```
RENAME B:ABODE HOME
```

renames the file ABODE on drive B to HOME.

The command:

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

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

SYS (System) Command

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

IBMBIO.COM
IBMDOS.COM

Format: SYS *d*:

Type:

Internal	External
	***

Remarks: The diskette in the specified drive must already be formatted by a FORMAT *d*:/S command to contain a copy of DOS. If you did not format the diskette with the /S option, the system cannot be transferred because the specific diskette locations required for the system files have not been allocated.

> **Note:** SYS lets you transfer a copy of DOS to an application program diskette designed to use DOS, but sold without it. In this case, the specific diskette locations required for the DOS files have already been allocated, although the DOS files are not actually present. The SYS command will transfer the files to the allocated space.

TIME Command

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.

Format: TIME

Type:

Internal	External
	***

Remarks: The TIME command issues the following prompt:

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

where:

hh is a one- or two-digit number from 0-23
(representing hours)
mm is a one- or two-digit number from 0-59
(representing minutes)
ss is a one- or two-digit number from 0-59
(representing seconds)
xx is a one- or two-digit number from 0-99
(representing hundredths of a second)

Notes:

1. To leave the time as is, press Enter.

2. If you enter any information (for example, just the hours, and press Enter), the remaining fields are set to zero.

3. Any time is acceptable as long as the digits are within the defined ranges.

4. The valid delimiters within the time are the colon (:) separating the hours, minutes, and seconds, and the period (.) separating the seconds and the hundredths of a second.

5. If you specify an invalid time or delimiter, you receive an **Invalid time** message.

Example: In this example, once you press Enter, the time known to the system is changed to 13:55:00.00.

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

TYPE Command

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

Format: TYPE *filespec*

Type:

Internal	External
***	

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

Notes:

1. Press Ctrl-PrtSc if you want the contents of a file to be printed as they are being displayed.

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

Example: In this example, the file **myprog.one** on the diskette in drive B is displayed on the screen.

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

Summary of DOS Commands

The following chart is provided for quick reference. The section called "Format Notation" at the beginning of this chapter explains the notation used in the format of the commands.

> **Note:** In the column labeled **Type**, the I stands for Internal and the E stands for External.

Command	**Type**	**Purpose**	**Format**
(Batch)	I	Executes batch file	[*d*:]*filename* [*parameters*]
CHKDSK	E	Checks disk and reports status	CHKDSK [*d*:]
COMP	E	Compares files	COMP [*filespec*] [*d*:] [*filename*[.*ext*]]
COPY	I	Copies files	COPY *filespec* [*d*:] [*filename*[.*ext*]]
DATE	E	Enter date	DATE
DIR	I	Lists filenames	DIR [*d*:] [*filename*[.*ext*]]
DISKCOMP	E	Compares diskettes	DISKCOMP [*d*:] [*d*:]
DISKCOPY	E	Copies diskettes	DISKCOPY [*d*:] [*d*:]
ERASE	I	Deletes files	ERASE *filespec*
FORMAT	E	Formats diskette	FORMAT [*d*:] [/S]

Command	**Type**	**Purpose**	**Format**
MODE	E	Sets mode on printer/display	MODE [LPT#:] [*n*] [,*m*] [,T]
PAUSE	I	Provides a system wait	PAUSE [*remark*]
REM	I	Displays remarks	REM [*remark*]
RENAME	I	Renames files	RENAME *filespec filename*[*.ext*]
SYS	E	Transfers DOS	SYS *d*:
TIME	E	Enter time	TIME
TYPE	I	Displays file contents	TYPE *filespec*

CHAPTER 4. THE LINE EDITOR (EDLIN)

Contents

Introduction 4-3

How to Start the EDLIN Program 4-4

The EDLIN Command Parameters 4-6

The EDLIN Commands 4-9
Information Common to All EDLIN Commands 4-9

Append Lines Command 4-11

Delete Lines Command 4-12

Edit Line Command 4-16

End Edit Command 4-18

Insert Lines Command 4-19

List Lines Command 4-22

Quit Edit Command 4-26

Replace Text Command 4-27

Search Text Command 4-30

Write Lines Command 4-33

Summary of EDLIN Commands 4-34

Introduction

In this chapter, you will learn how to use the Line Editor (EDLIN) program.

You can use the Line Editor (EDLIN) to create, change, and display *source files* or text files. Source files are unassembled programs in source language format. Text files appear in a legible format.

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

- Create new source files and save them.
- Update existing files and save both the 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 you insert lines, all line numbers following the inserted text advance automatically by the number of lines inserted. When you delete lines, all line numbers following the deleted text decrease automatically by the number of lines deleted. Consequently, line numbers always go consecutively from 1 through the last line.

How to Start the EDLIN Program

To start EDLIN, enter:

```
EDLIN filespec
```

- If the specified file exists on the designated or default drive, the file is loaded into memory until memory is 75% full. If the entire file is loaded, the following message and prompt is displayed:

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

  You can then edit a file.

  Notice that the prompt for EDLIN is an asterisk (*).

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

  To edit the remainder of the file, you must write some of the edited lines to diskette in order to free memory so that you can load unedited lines from diskette into memory. Refer to the Write Lines and Append Lines commands in this chapter for the procedure you will use.

- If the specified file does not exist on the drive, a new file is created with the specified name. The following message and prompt is displayed:

```
New file
*_
```

  You can now create a new file by entering the desired lines of text. To begin entering text, you must enter an **I** command to insert lines.

When you have completed the editing session, you can save the original and updated (new) files by using the End Edit command. The End Edit command is discussed in this chapter in the section called "The EDLIN Commands." The original file is renamed to an extension of .BAK, and the new file has the filename and extension you specified in the EDLIN command.

Notes:

1. You cannot edit a file with a filename extension of .BAK with EDLIN because the system assumes it is a backup file. If you find it necessary to edit such a file, rename the file to another extension; then start EDLIN and specify the new name.

2. When you start EDLIN, EDLIN erases the backup copy (.BAK) of the file, if one exists, to ensure adequate space on the diskette for the updated file. EDLIN then allocates a new file with the filename that you specify in the EDLIN command and an extension of .$$$. The file with an extension of $$$ contains the updated file, and is ultimately renamed to the specified filespec by the End Edit command.

The EDLIN Command Parameters

Parameter	Definition
line	Denotes when you must specify a line number. There are three possible entries that you can make using this parameter: 1. Enter a decimal integer from 1-65529. If you specify a number greater than the number of lines that are in memory, the line will be added after the last line that exists. Line numbers must be separated from each other by a comma or a space. OR 2. Enter a pound sign (#) to specify the line after the last line in memory. Entering a # has the same effect as specifying a number greater than the number of lines in memory.

Parameter	Definition
line	OR 3. Enter a period (.) to specify the current line. The current line indicates the location of the last change to the file, but is not necessarily the last line displayed. The current line is marked by an asterisk (*) between the line number and the first character of text in the line. For example: 10:*FIRST CHARACTER OF TEXT
n	Denotes when you must specify lines. Enter the number of lines that you want to write to diskette or load from diskette. You only use this parameter 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.

Parameter	Definition
string	Denotes when you must enter one or more characters to represent text to be found, replaced, deleted; or to replace other text. You only use this parameter with the Search Text and Replace Text commands.

The EDLIN Commands

This section describes the EDLIN commands and tells how to use them. The commands are in alphabetical order; each with its purpose and format. Examples are provided where appropriate.

Information Common to All EDLIN Commands

The following information applies to all EDLIN commands:

- With the exception of the Edit Line command, all commands are a single letter.
- With the exception of the End Edit and Quit Edit commands, commands are usually preceded and/or followed by parameters.
- Enter commands and string parameters in uppercase or lowercase, or a combination of both.
- Separate commands and parameters by delimiters for readability; however, a delimiter is only required between two adjacent line numbers. Remember, delimiters are spaces or commas.
- Commands become effective only after you press the Enter key.
- End commands by pressing the Ctrl-Break keys.
- For commands producing a large amount of output, press Ctrl-NumLock to suspend the display so that you can read it before it scrolls away. Press any other character to restart the display.

- Use the control keys and DOS editing keys, described in Chapter 1, while using EDLIN. They are very useful for editing *within a line*, while the EDLIN commands can be used for editing operations on *entire lines.*

- The prompt from EDLIN is an asterisk (*).

Append Lines Command

Purpose: Adds the specified number of lines from diskette to the file being edited in memory. The lines are added at the end of the current lines in memory.

Format: [*n*] A

Remarks: This command is only meaningful if the file being edited is too large to fit in memory. As many lines as possible are read into memory for editing when you start EDLIN.

To edit the remainder of the file that will not fit into memory, you must write edited lines in memory to diskette before you can load unedited lines from diskette into memory by using the Append Lines command. Refer to the Write Lines command for information on how to write edited lines to diskette.

Notes:

1. If you do not specify the number of lines, lines are appended to memory until available memory is 75% full. No action is taken if available memory is already 75% full.

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

Delete Lines Command

Purpose: Deletes a specified range of lines.

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

Remarks: 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.

Default values are supplied if either one or both of the parameters are omitted.

If you omit the first parameter, as in:

,*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 you omit the second parameter, as in:

*line*D

or

line,D

only the one specified line is deleted.

If you omit both parameters, as in:

```
D
```

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

Example: Assume that you want to edit the following file. The current line is line 29.

```
 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.
```

If you want to delete a range of lines, from 5-25, enter:

```
5,25 D
```

The result is:

```
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. Line 5 becomes the current line.

If you want to delete the current and the following line, enter:

```
,6 D
```

The result is:

```
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. Lines 7-8 are renumbered to 5-6. Line 5 is still the current line, but now it has different text.

If you want to delete a single line, say line 2, enter:

```
2 D
```

The result is:

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

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

If you want to delete only the current line, enter:

```
D
```

The result is:

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

The current line, line 2, is deleted. Lines 3-5 are renumbered to 2-4. The new line 2 becomes the current line.

Edit Line Command

Purpose: Allows you to edit a line of text. You must enter the line number of the line to be edited, or enter a period (.) to indicate the current line.

Format: [*line*]

Remarks: If you just press Enter, you specify that the line after the current line is to be edited.

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

You can use the control keys and the editing keys, described in Chapter 1, to edit the line, or you can replace the entire line by typing new text.

When you press the Enter key, the edited line is placed in the file and becomes the current line.

If you decide not to save the changed line, either press Esc or press Ctrl-Break. The original line remains unchanged. Pressing the Enter key with the cursor at the beginning of the line has the same effect as pressing Esc or Ctrl-Break.

If the cursor is in any position other than the beginning or the end of a line, pressing Enter truncates the rest of the line.

Example: Assume that you want to edit line 6. The following display would appear on the screen:

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

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

If you want to move the cursor to the letter **u**, press F2 and enter:

```
u
```

The result is:

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

If you want to delete the next two characters and keep the remainder of the line, press Del twice; then press F3.

The result is:

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

Now you can take one of the following 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 additional editing to the changed line without changing the original line.

- Press Esc or Ctrl-Break to cancel the changes you made to the line. The original contents of the line will be preserved.

End Edit Command

Purpose: Ends EDLIN and saves the edited file.

Format: E

Remarks: The edited file is saved by writing it to the drive and filename specified when you started EDLIN.

The original file, the one specified when EDLIN was started, is renamed by giving it a .BAK filename extension. A .BAK file will not be created if there is no original file; that is, if you created a new file instead of updating an old file during the editing session.

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

> **Note:** Be sure your diskette has enough free space to save the entire file. If your diskette does not have enough free space, only a portion of the file is saved. The portion in memory that is not written to diskette is lost.

Insert Lines Command

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

Format: [*line*] I

Remarks: If you do not specify a line, or if you specify line as a period (.), the insert is made immediately before the current line.

If the line number you specify is greater than the highest existing line number, or if you specify # as the line number, the insertion is made after the last line in memory.

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

You must press Ctrl-Break to discontinue the insert mode of operation.

The line that follows the inserted lines becomes the current line, even if the inserted lines are added to the end of the lines in memory. The current line and all the remaining lines are renumbered.

When you create a new file, you must enter the Insert Lines command before text can be inserted.

Insert Lines Command

Example: Assume that you want to edit the following file. 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.
```

If you want to insert text before line 4, the entry and immediate response looks like this:

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

Now, if you want to insert two new lines of text, enter:

```
*4   I
     4:*First new line of text
     5:*Second new line of text
     6:*
```

and press Ctrl-Break.

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

If you display the file with a List Lines command, the file looks like this:

```
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 lines that were inserted had been placed at the beginning of the file, the screen would look like this:

```
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 lines that were inserted had been placed immediately before the current line (3 I or . I or I), the screen would look like this:

```
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 look like this:

```
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

Purpose: Displays a specified range of lines.

The current line remains unchanged.

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

Remarks: Default values are provided if either one or both of the parameters are omitted.

If you omit the first parameter, as in:

,line L

the display starts 11 lines before the current line and ends with the specified line. The beginning comma is required to indicate the omitted first parameter.

> **Note:** If the specified line is more than 11 lines before the current line, the display is the same as if you omitted both parameters. (An example is provided in this section showing both parameters omitted.)

If you omit the second parameter, as in:

line L

or

line, L

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

If you omit both parameters, as in:

```
L
```

a total of 23 lines are 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, the extra lines after the current line are displayed to make a total of 23 lines.

Example: Assume that you want to edit the following file. 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.
```

If you want to display a range of lines, from 5-25, enter:

```
5,25 L
```

The screen looks like this:

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

If you want to display the first three lines, enter:

```
1,3 L
```

The screen looks like this:

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

If you want to display 23 lines of the file, starting with line 3, enter:

```
3 L
```

The screen looks like this:

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

If you want to display 23 lines centered around the current line, enter:

```
L
```

The screen looks like this:

```
 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

Purpose: Quits the editing session without saving any changes you may have entered.

Format: Q

Remarks: EDLIN prompts you to make sure you really don't want to save the changes.

Enter Y if you want to quit the editing session. No editing changes are saved and no .BAK file is created. Refer to the End Edit command for information about the .BAK file.

Enter N, or any other character, if you want to continue the editing session.

> **Note:** When started, EDLIN erases any previous backup copy of the file (filename.BAK) to make room for saving the new copy. Therefore, if you reply Y to the **Abort edit (Y/N)?** message, your previous backup copy will no longer exist.

Example:

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

Replace Text Command

Purpose: Replaces all occurrences of the first string in the specified range of lines with the second string.

> **Note:** If you omit the second string, Replace Text deletes all occurrences of the first string within the specified range of lines.

Displays the changed lines each time they are changed. The last line changed becomes the current line.

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

Remarks: You can specify the optional parameter ? to request a prompt (**O.K.?**) after each display of a modified line. Press the Y, or the Enter key if you want to keep the modification.

Enter any other character if you don't want 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.

Defaults occur if either one or both of the parameters is missing.

If you omit the first line, the search begins with line 1. If you omit the second line, the search ends with the last line in memory. If you omit both line parameters, the system will 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 you press F6 or Ctrl-Z (or by pressing the Enter key if the second string is omitted). If you omit the first string, no search can be made, so the command ends immediately and the **Not found** message is displayed.

The second string begins immediately after you press F6 or Ctrl-Z and continues until you press Enter.

Example: Assume that you want to edit the following file. 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
```

Then press F6 or Enter.

The result is:

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

Line 6 becomes the current line in the file because line 6 was the last line changed. Notice that 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 using the ? parameter) after each display of a modified line. If you request a prompt, the screen looks like this:

```
*1,7?Rand    (Press 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 are displayed like this:

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

Search Text Command

Purpose: Searches a specified range of lines in order to locate a specified string.

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

Remarks: The first line to contain the specified string is displayed and the search ends (unless you use the ? parameter). The first line found that matches the specified string becomes the current line.

You should specify the optional parameter ? if you would like a prompt (O.K.?) after each display of a line containing the specified string.

If the specified string is not found, the search ends and the message **Not found** is displayed. The current line remains unchanged.

If you press the Enter key or the Y key, the line that matches the specified string becomes the current line and the search ends. Enter any other character to continue the search until another string is found, or until all lines within the range are searched. Once all the lines within the range are searched, the **Not found** message is displayed.

The system provides default values if you omit the first, second, or both line parameters. If you omit the first line parameter, the system defaults to line 1. If you omit the second line parameter, the system defaults to the last line in your file. If you omit both line parameters, the system searches all lines in memory.

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

Example: Assume that you want to edit the following file. 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.
```

If you want 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 is:

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

The **and** is part of the word **command**. Notice that line 3 becomes the current line in the file.

Perhaps this is not the **and** you were looking for. To continue the search; enter a new starting line number (4); press F3 to copy the remainder of the previous S command to the screen; then press Enter to execute the S command starting at line 4.

The screen looks like this:

```
*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 search for strings by requesting a prompt (by means of the ? parameter) after each display of a matching line. In this case, the screen looks like this:

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

Write Lines Command

Purpose: Writes a specified number of lines to diskette from the lines that are being edited in memory. Lines are written beginning with line number 1.

Format: [*n*] W

Remarks: This command is only meaningful if the file you are editing is too large to fit in memory. When you start EDLIN, EDLIN reads lines into memory until memory is 75% full.

To edit the remainder of your file, you must write edited lines in memory to diskette before you can load additional unedited lines from diskette into memory by using the Append Lines command.

> **Note:** If you do not specify the number of lines, lines are written until 25% of available memory is 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.

Summary of EDLIN Commands

The following chart is provided for quick reference.

Note: The section called "Format Notation" in Chapter 3 explains 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

CHAPTER 5. THE LINKER (LINK) PROGRAM

Contents

Introduction 5-3

Files 5-4
Input Files 5-4
Output Files 5-4
VM.TMP (Temporary File) 5-5

Definitions 5-6
Segment 5-6
Group 5-7
Class 5-7

Command Prompts 5-8

Detailed Descriptions of the Command Prompts 5-10
Object Modules: 5-10
Run File: 5-11
List File [run-filename.MAP]: 5-11
Libraries [ ]: 5-12
Publics [No]: 5-13
Line Numbers [No]: 5-13
Stack Size [Object file stack]: 5-13
Load Low [Yes]: 5-14
DSAllocation [No]: 5-15

Special Command Characters 5-16

How to Start LINK 5-17
Before You Begin 5-17
Example of an Automatic Response File . . 5-19
Example Linker Session 5-20
Load Module Memory Map 5-24
How to Determine the Absolute Address
of a Segment 5-25
Messages 5-26

Introduction

The Linker (LINK) program 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.

In this chapter, you will learn how to start LINK at the end of the chapter. You should read all of this chapter before you start LINK.

Files

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

Input Files

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

Output Files

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

VM.TMP (Temporary Files)

LINK uses as much memory as is available to hold the data that defines the load module being created. If the module is too large to be processed with the available amount of memory, the linker may need additional memory space. If this happens, a temporary diskette file called VM.TMP is created on the DOS default drive.

A message is displayed to indicate when the overflow to diskette has begun. Once this temporary file is created, you should not remove the diskette 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 file will be allocated. The contents of the previous file are destroyed; therefore, you should avoid using VM.TMP as one of your own filenames.

Definitions

Segment, *group*, and *class* are terms that appear in this chapter and in some of the 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 when machine language is generated.

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 Load Low prompt will be discussed in this chapter.

Group

A *group* is a collection of segments that fit together within a 64K-byte segment of memory. The segments are named to the group by the assembler or compiler. A program may consist of one or more groups.

The group is used for addressing segments in memory. The various portions of segments within the group are addressed by a segment base pointer plus an offset. The linker checks that the object modules of a group meet the 64K-byte constraint.

Class

A *class* is a collection of segments. The naming of segments to a class affects the order and relative placement of segments in memory. The class name is specified by the assembler or compiler. All portions assigned to the same class name are loaded into memory contiguously.

The segments are ordered within a class in the order that the linker encounters the segments in the object files. One class precedes another in memory only if a segment for the first class precedes all segments for the second class in the input to LINK. Classes are not restricted in size. The classes will be divided into groups for addressing.

Command Prompts

After you start the linker session, you receive a series of nine prompts. You can respond to these prompts from the keyboard, or you can use a special diskette file that is called an *automatic response file* to respond to the prompts. An example of an automatic response file is provided in this chapter. Refer to the section called "How to Start LINK" in this chapter for information on how to start the Linker session.

LINK prompts you for the names of object, run, list, and library files. LINK also prompts you 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 will display a message.

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

Prompt	**Responses**
Object Modules:	*filespec*[*filespec*. . .]
Run File:	*filespec* [/P]
List File [filename.MAP] :	[*filespec*]
Libraries [ ]:	[*filespec*[*filespec*. . .]]

Prompt	Responses
Publics [No] :	Y or N: Y lists all global symbols with definitions.
Line Numbers [No] :	Y or N: Y includes line numbers in LIST file.
Stack Size [Object file stack] :	Non-zero decimal value sets stack size in RUN file.
Load Low [Yes] :	Y or N: Y causes RUN file to be loaded in low memory when executed.
DSAllocation [No] :	Y or N: Y loads data at high-end of data segment.

Notes:

1. For the prompts that require a **Yes** or a **No**, only the first character of the response is interpreted. The first character must either be a Y or an N. Any additional characters that you may provide are ignored by the linker. You can also press the Enter key instead of responding with a Y or an N. If you press Enter, default values are provided. The default values that are provided will be discussed in the detailed descriptions of the command prompts in this chapter.

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

3. You can end the linker session prior to its normal end by pressing Ctrl-Break.

Detailed Descriptions of the Command Prompts

The following detailed descriptions contain information about the responses that you can enter to the prompts.

Object Modules:

Enter one or more filespecs for the object modules to be linked. If the extension is omitted, LINK assumes the filename extension .OBJ. If an object module has another 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 you specify an object module, but LINK cannot locate the file, a prompt requests you to insert the diskette containing the specific module. This permits .OBJ files from several diskettes to be included.

To avoid a conflict with VM.TMP (which may be 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. The VM.TMP file is discussed in this chapter.

IMPORTANT: If a VM.TMP file has been opened, you should *not* remove the diskette containing the VM.TMP file. Remember, once a VM.TMP file is opened, the diskette it resides on cannot be removed.

If a VM.TMP file has been opened and you receive a prompt to insert another diskette, you must either specify another drive whose diskette can safely be exchanged, or press the Ctrl-Break keys. Pressing Ctrl-Break allows you to end LINK. Then you can set up your files to allow space for the VM.TMP file *before* rerunning LINK.

Run File:

The filespec you enter is 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. 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 you. This message will request you to insert the diskette that is to receive the Run file. If you use the /P option, 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. If the Run file is not directed to the default drive (where VM.TMP is allocated), you should send the List file to either the default drive, or to a drive whose diskette is not going to be removed. If the diskette that the List file is being sent to is removed in response to any of the prompts requesting diskette changes, the directory of the replacement diskette is invalid—which causes any files it may have contained to be lost.

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.

The Run filename with the default extension .MAP is used if you do not enter a filespec.

> **Note:** The diskette that the .MAP file is allocated to must not be removed until the LINK has ended. If a prompt requests that you remove the .MAP diskette; end the LINK by pressing the Ctrl-Break keys; then reorganize the files needed for the LINK on the appropriate diskettes.
>
> To avoid generating the .MAP file on a diskette, you can specify the console as the List file device. For example:
>
> ```
> List File [run-filename.MAP]: CON
> ```
>
> If you direct the output to your console, you can also print a copy of the output by pressing the Ctrl-PrtSc keys.

Libraries [ ]:

The valid responses are either listing the library filespecs, or pressing the Enter key. If you just press the Enter key, LINK defaults to the library provided as part of the Compiler package. For linking objects from just the Assembler, there is no automatic default library search.

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

> **Note:** For PASCAL, LINK already knows the library name; therefore, you do not need to specify one.

You can enter from 1-8 library filespecs. The filespecs must be separated by single commas or spaces.

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

Publics [No]:

The only valid responses are Y and N, or you can simply press 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 [No]:

The only valid responses are Y and N, or you can simply press the Enter key. Pressing only the Enter key defaults to N.

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

For modules produced by the Assembler, the response is assumed to be N.

Stack Size [Object file stack]:

Valid responses are any positive decimal value up to 65536 bytes, or you can simply press the Enter key. If your response is 0, or if you just press the Enter key, you specify that the original stack size provided by the assembler or compiler is to be used.

If you specify a value greater than 0 but less than 512, 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.

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 [Yes]:

The only valid responses are Y and N, or you can simply press the Enter key. Pressing only the Enter key defaults to Y.

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.

DSAllocation [No]:

The only valid responses are Y and N, or you can simply press 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 the 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 only 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.

Special Command Characters

LINK recognizes two special command characters:

Character	Definition
&	Use the ampersand (&) to extend the current line. To enter a large number of responses (each of which can 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 pressing 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, only use the Enter key.

How to Start LINK

Before You Begin

- Make sure the files you will be using for the LINK are on the appropriate diskettes.
- Make sure you have enough free space on your diskettes to contain your files and any generated data.

You can start the Linker 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. Detailed descriptions of the responses that you can enter to the prompts are discussed in this chapter in the section called "Command prompts."

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 you must restart LINK. If the error has been typed but not entered, you may delete the erroneous characters, for that line only.

An example of a Linker session, using the console response option, is provided in this chapter in the section called "Example Linker Session."

Option 2–Automatic Responses

From your keyboard, enter:

```
LINK filespec
```

For this option, you enter the filespec of an Automatic Response File. An example of an Automatic Response File is provided in this chapter.

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 that were discussed earlier in this chapter. 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 command that starts LINK to be entered from the keyboard or within a batch file without requiring any response from you.

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

1. The text contained within the brackets is treated as a prompting remark that 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 waiting for a response from the keyboard. You may then enter additional items (ending them by pressing the Enter key), or you may just press the Enter key. The linker continues processing with the next statement from the Automatic Response File.

Example of an Automatic Response File

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
PASCALLIB
Y
Y
C
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 when you press the Enter key.

LINK

Example Linker Session

This example shows you the type of information that is displayed during a linker session.

Once you enter:

```
A>b:link
```

the system responds with the following messages:

```
IBM Personal Computer Linker
Version 1.00 (C) Copyright IBM Corp 1981
Object Modules: example
Run File: example
List File [EXAMPLE.MAP]:prn
Libraries [   ]:
Publics [No]: y
Line Numbers [No]: y
Stack size [Object file stack]:
Load Low [Yes]:
DSAllocation [No]:
```

Notes:

1. By responding **prn** to the List file prompt, we sent our output to the printer.

2. By just pressing Enter in response to the Libraries prompt, an automatic library search is performed.

3. By responding **y** to the Publics prompt, we get both an alphabetic listing and a chronological listing of public symbols.

4. By responding **y** to the Line Numbers prompt, LINK gives us a listing of all line numbers for all modules. (A **y** response to the Line Numbers prompt can generate a large amount of output.)

If LINK cannot locate a library on the specified drive, the following message is displayed:

```
cannot find library A:PASCAL.LIB
enter new drive letter:
```

The drive that the indicated library is located on must be entered.

Once LINK locates all libraries, the Linker .MAP displays a list of segments in the relative order of their appearance within the load module. The list looks like this:

```
Start   Stop    Length Name           Class

00000H  00028H  0029H  MAINQQ         CODE
00030H  000F6H  00C7H  ENTXQQ         CODE
00100H  00100H  0000H  INIXQQ         CODE
00100H  038D3H  37D4H  FILVQQ_CODE    CODE
038D4H  04921H  104EH  FILUQQ_CODE    CODE
•
•
•
074A0H  074A0H  0000H  HEAP           MEMORY
074A0H  074A0H  0000H  MEMORY         MEMORY
074A0H  0759FH  0100H  STACK          STACK
075A0H  07925H  0386H  DATA           DATA
07930H  082A9H  097AH  CONST          CONST
```

The information in the **Start** and **Stop** columns shows a 20-bit hex address of each segment relative to location zero. Location zero is the beginning of the load module. The addresses displayed are not the absolute addresses of where these segments are loaded. To find the absolute address of where a segment is actually loaded, you must determine where the segment listed as being at relative zero is actually loaded; then add the absolute address to the relative address shown in the .MAP listing. The procedure you use to determine where relative zero is actually located is discussed in this chapter, in the section called "How to Determine the Absolute Address of a Segment."

Now, because we responded **y** to the Publics prompt, the public symbols are displayed by name and by value. For example:

```
Address          Publics by Name

0492:0003H       ABSNQQ
06CD:029FH       ABSRQQ
0492:00A3H       ADDNQQ
06CD:0087H       ADDRQQ
0602:000FH       ALLHQQ
.
.
.
0010:1BCEH       WT4VQQ
0010:1D7EH       WTFVQQ
0010:1887H       WTIVQQ
0010:19E2H       WTNVQQ
0010:11B2H       WTRVQQ

Address          Publics by Value

0000:0001H       MAIN
0000:0010H       ENTGQQ
0000:0010H       MAINQQ
0003:0000H       BEGXQQ
0003:0095H       ENDXQQ
.
.
.
F82B:F31CH       CRCXQQ
F82B:F31EH       CRDXQQ
F82B:F322H       CESXQQ
F82B:F5B8H       FNSUQQ
F82B:F5E0H       OUTUQQ
```

The addresses of the public symbols are also in the *segment:offset* format, showing the location relative to zero as the beginning of the load module. In some cases, an entry may look like this:

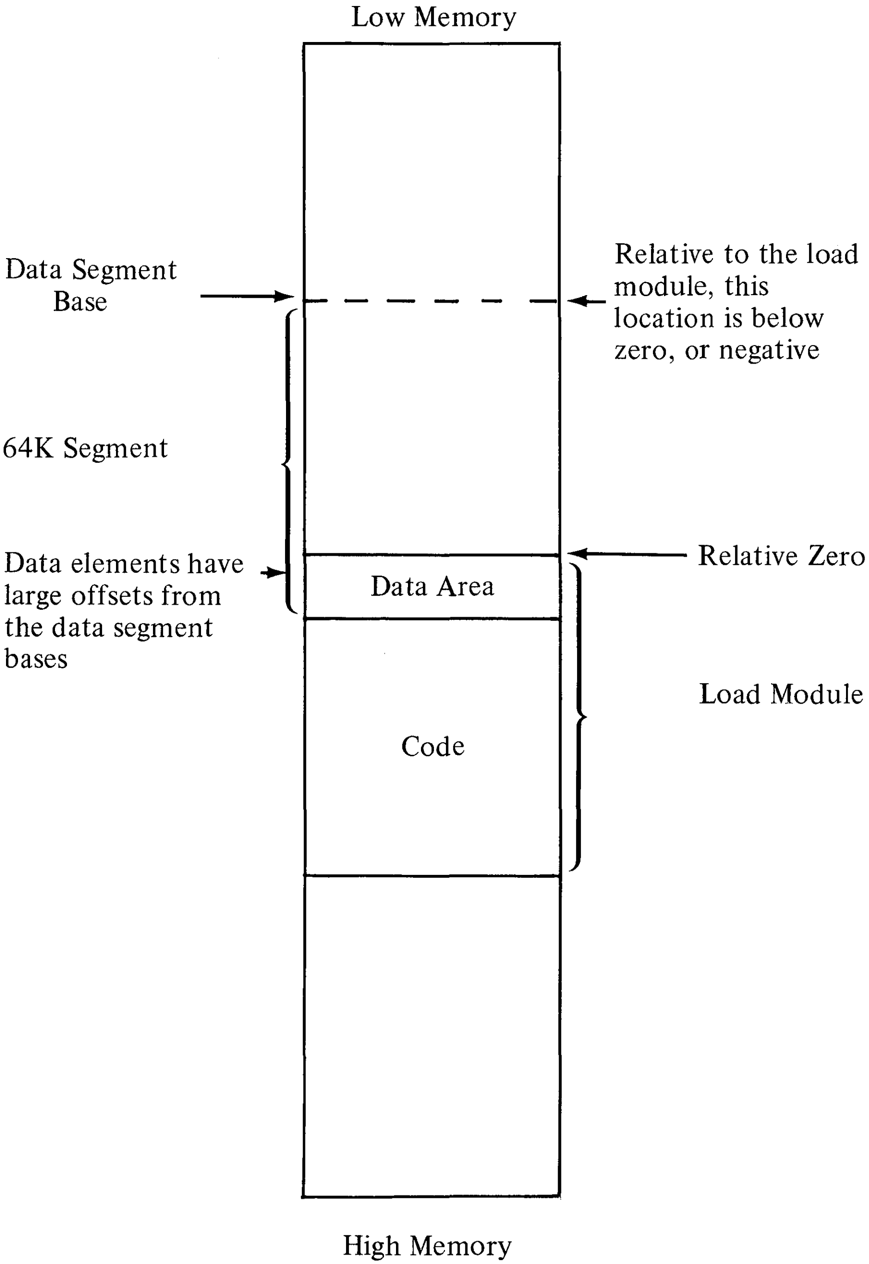
```
F0C0:E924
```

This entry appears to be the address of a load module that is almost one megabyte in size. Actually, the area being referenced is relative to a segment base that is pointing to a segment below the relative zero beginning of the load module. This condition produces a pointer that has effectively gone negative. The chart on the following page is provided to illustrate this point.

When LINK has completed, the following message is displayed:

```
Program entry point at 0003:0000
```

Load Module Memory Map


Low Memory
Data Segment Base
Relative to the load module, this location is below zero, or negative
64K Segment
Data elements have large offsets from the data segment bases
Relative Zero
Data Area
Load Module
Code
High Memory

How to Determine the Absolute Address of a Segment

The Linker .MAP displays a list of segments in the relative order of their appearance within the load module. The information displayed shows a 20-bit hex address of each segment relative to location zero. The addresses that are displayed are not the absolute addresses of where these segments are actually located. To determine where relative zero is actually located, we must use DEBUG. DEBUG is discussed in Chapter 6.

Using DEBUG,

1. Load the application.

   Note the segment value in CS and the offset within that segment to the entry point as shown in IP. The last line of the Linker .MAP also describes this entry point, but uses relative values, not the absolute values shown by CS and IP.

2. Subtract the relative entry as shown at the end of the .MAP listing from the CS:IP value. For example, let's say CS is at 05DC and IP is at zero.

   The Linker .MAP shows the entry point at 0100:0000. (0100 is a segment ID or paragraph number; 0000 is the offset into that segment.)

   In this example, relative zero is located at 04DC:0000, which is 04DC0 absolute.

If a program is loaded low, the relative zero location is located at the end of the Program Segment Prefix, or in the value in DS plus 100H.

Messages

All messages, except for the warning messages, cause the LINK session to end. Therefore, after you locate and correct a problem, you must rerun LINK.

Refer to Appendix A for a complete listing of messages.

CHAPTER 6. THE DEBUG PROGRAM

Contents

Introduction 6-3

How to Start the DEBUG Program 6-4

The DEBUG Command Parameters 6-6

The DEBUG Commands 6-10
Information Common to All DEBUG Commands 6-10

Dump Command 6-12

Enter Command 6-15

Fill Command 6-18

Go Command 6-19

Hexarithmetic Command 6-22

Input Command 6-23

Load Command 6-24

Move Command 6-27

Name Command 6-28

Output Command 6-30

Quit Command 6-31

Register Command 6-32

Search Command 6-37

Trace Command 6-38

Unassemble Command 6-40

Write Command 6-43

Summary of DEBUG Commands 6-47

Introduction

This chapter explains how to use the DEBUG program.

The DEBUG program can be used to:

- Provide a controlled test 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 the program immediately to determine if the problems have been resolved. You do not need to reassemble a program to find out if your changes worked.

- Load, alter, or display any file.

- Execute *object files*. Object files are executable programs in machine language format.

How to Start the DEBUG Program

To start DEBUG, enter:

```
DEBUG [filespec]
```

If you enter a 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 a filespec, you must either work with the present memory contents, or load the required file into memory by using 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 start 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. (Refer to 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. The DS and ES registers, however, 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. Refer to “.EXE File Structure and Loading” in Appendix F for more information about loading .EXE files.

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.

The DEBUG Command Parameters

Parameter	Definition
address	Enter a one- or two-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: 100 (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 nonexistent memory location.
byte	Enter a one or two character *hexadecimal* value.

Parameter	Definition
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. (Refer to the Load and Write commands.)
filespec	Enter a one- to three-part file specification consisting of a drive designation, filename, and filename extension. All three fields are optional. However, for the Name command to be meaningful, you should at least specify a drive designator or a filename. (Refer to the Name command.)
list	Enter one or more byte and/or string values. For example: E CS:100 F3 'XYZ' 8D 4 "abcd" has five 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. (Refer to the Input and Output commands.)
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 a comma.

Parameter	Definition
range	● *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 four *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.
registername	Refer to the Register command.
sector sector	Enter 1-3 character *hexadecimal* values to specify the starting relative sector number and 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. (Refer to the Load and Write commands.)
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.

Parameter	**Definition**
string	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. 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 second and sixth cases above, the word *literal* is enclosed in one set of quotation marks in memory. In the fourth and eighth 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 (refer to 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. (Refer to 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 and purpose. Examples are provided where appropriate.

Information Common to All DEBUG Commands

The following 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 uppercase or lowercase, or a combination of both.

- Commands and parameters may be separated by delimiters. 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
  ```

- Press Ctrl-Break to end commands.

- Commands become effective only after you press the Enter key.

- 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 character to restart the display.

- You can use the control keys and the DOS editing keys, described in Chapter 1, 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

Purpose: Displays the contents of a portion of memory.

Format: D [*address*]

or

D [*range*]

Remarks: The dump is displayed in two parts:

1. A hexadecimal portion. Each byte is displayed in hexadecimal.
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.

Dump Command

The Dump command has two format options:

Option 1

Use this option to display the contents of X‘40’ bytes (40-column mode) or X‘80’ bytes (80-column mode). For example:

D *address*

or

D

The contents are dumped starting with the specified address.

If you do not specify an address, 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

Use this option to display the contents of the specified address range. For example:

D *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 10C
```

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 47
                      BORG
```

Enter Command

Purpose: The Enter command has two modes of operation:

- Replaces the contents of one or more bytes, starting at the specified address, with the values contained in the list. (See Option 1.)

- Displays and allows modification of bytes in a sequential manner. (See Option 2.)

Format: E *address* [*list*]

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

The Enter command has two format options:

Option 1

Use this option to place the list in memory beginning at the specified address.

E *address* *list*

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.

Option 2

Use this option to display the address and the byte of a location, then the system waits for your input.

For example:

E *address*

Now you can take one of the following actions:

1. Enter a one or two 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 the 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 the 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

Purpose: Fills the memory locations in the range with the values in the list.

Format: F *range list*

Remarks: 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

Purpose: Executes the program you are debugging.

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.

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

Remarks: 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*.

The Go command has two format options:

Option 1

Use this option to execute the program you are debugging without breakpoints. For example:

G [=*address*]

This option is useful when testing program execution with different parameters each time. (Refer to the Name command.) Be certain the CS:IP values are set properly before issuing the G command, if not using =*address*.

Option 2

This option performs the same function as Option 1 but, in addition, allows breakpoints to be set at the specified addresses. For example:

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

This method causes execution to stop at a specified location so the system/program environment can be examined.

You can specify up to ten breakpoints 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. Make sure that the address parameters refer to locations that contain valid 8088 instruction codes. If you specify an address that does not contain the first byte valid instruction, unpredictable results will occur.

2. The stack pointer must be valid and have 6 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:

```
[illegible]
```

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, all three breakpoints are removed, the display occurs, and the Go command ends.

Refer to the Register command for a description of the display.

Hexarithmetic Command

Purpose: Adds the two hexadecimal values, then subtracts the second from the first.

Displays the sum and difference on one line.

Format: H *value value*

Example:

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

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

Input Command

Purpose: Inputs and displays (in hexadecimal) one byte from the specified port.

Format: I *portaddress*

Example:

```
I 2F8
6B
```

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

Load Command

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

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

Remarks: The maximum number of sectors that can be loaded with a single Load command is X‘80’.

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

The Load command has two format options:

Option 1

Use this option to load data from the diskette specified by drive, and place the data in memory beginning at the specified address. For example:

L *address drive sector sector*

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 you only enter an offset 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. X'6D' (109) consecutive sectors of data are transferred, starting with relative sector X'0F' (15) (the 16th sector on the diskette).

Option 2

When issued without parameters, or with only the address parameter, use this option to load the file whose filespec is properly formatted in the file control block at CS:5C. For example:

L

or

L *address*

This condition is met by specifying the filespec when starting the DEBUG program, or by using the Name command.

> **Note:** If DEBUG was started 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; however, if the file being loaded has an extension of .EXE, CX is set to zero and the file may be loaded at the high end of memory. Refer to the notes in "How to Start the DEBUG Program" at the beginning of this chapter for the conditions that are in effect when .EXE or .HEX files are loaded.

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

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

Format: M *range address*

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

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

Notes:

1. If you only enter an offset 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 you only enter an offset 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 17 bytes of data from CS:100 through CS:110 are moved to the area of memory beginning at DS:500.

Name Command

Purpose: The Name command has two functions:

- Formats file control blocks for the first two filespecs, at CS:5C and CS:6C. (Starting 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.

- 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.

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

Remarks: If you start the DEBUG program without a filespec, you must use the Name command before a file can be loaded with the L 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

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

Format: O *portaddress byte*

Example: To send the byte value 4F to output port 2F8, enter:

```
O 2F8 4F
```

Quit Command

Purpose: Ends the DEBUG program.

Format: Q

Remarks: The file that you are working on in memory is *not* saved by the Quit command. You must use the Write command to save the file.

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

Example:

```
-Q
A>
```

Register Command

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.

Format: R [*registername*]

Remarks: When the DEBUG program starts, the registers and flags are set to certain values for the program being debugged. (Refer to "How to Start the DEBUG Program" at the beginning of this chapter.)

Display a Single Register

The valid **registernames** are:

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

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
```

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

To display all flags, enter:

```
R F
```

If all the flags are in a *set* condition, the response is:

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

Now you can take one of two actions:

1. Press Enter to leave the settings unchanged.
2. Change any or all of the settings.

To change a flag, just enter its opposite code. The opposite codes can be entered in any order–with or without intervening spaces. For example, to change the first, third, fifth, and seventh 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 if the new codes are in effect, enter:

```
R F
```

The response will be:

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

The first, third, fifth, and seventh flags are changed as requested. The second, fourth, sixth, and eighth flags are unchanged.

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

Search Command

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

Format: S *range list*

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

A display of the prompt (-) without an address 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: If you want to search the range of addresses from CS:100 through CS:110 for X‘41’, enter:

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

If two matches are found the response might be:

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

If you want to search the same range of addresses as in the previous example for a match with the 4-byte-long list, enter:

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

The starting addresses of all matches are listed. If no match is found, no address is displayed.

Trace Command

Purpose: Executes 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*.

Displays the contents of all registers and flags *after each* instruction executes. For a description of the display format, refer to the Register command.

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

Remarks: The display caused by the Trace command continues until value instructions are executed. Therefore, when tracing multiple instructions, remember 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 displayed:

```
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=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
```

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 you suspend the display by pressing the Ctrl-NumLock keys.

Unassemble Command

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

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

Format: U [*address*]

or

U [*range*]

Remarks: 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. Make sure that the address parameters refer to locations containing valid 8088 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.

The Unassemble command has two format options:

Option 1

Use this option to either unassemble instructions without specifying an address, or to unassemble instructions beginning with a specified address. For example:

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 you do not specify an address, 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 is entered, the location is offset X'0100' into the segment originally initialized in the segment registers by DEBUG.

Option 2

Use this option to unassemble instructions in a specified address range. For example:

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.

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 appears if you enter:

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

or

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

or

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

Write Command

Purpose: Writes the data being debugged to diskette.

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

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

DEBUG displays a message if a diskette write error occurs. The write operation can be retried by pressing F3 to re-display the Write command, then press the Enter key.

The Write command has two format options:

Option 1

Use this option to write data to diskette beginning at a specified address. For example:

```
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 you write 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.

3. Remember, the starting sector and the sector count are both specified in *hexadecimal.*

For example:

```
W 1FD 1 100 A
```

The data beginning at CS:01FD is written to the diskette in drive B, starting at relative sector X'100' (256) and continuing for X'0A' (10) sectors.

Option 2

This option allows you to use the Write command without specifying parameters or only specifying the address parameter. For example:

W

or

W *address*

When issued without parameters (or when issued with only the address parameter), the Write 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 starting the DEBUG program, or by using the Name command.

> **Note:** If DEBUG was started 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 at the location specified by address, is written to the diskette in the drive specified in filespec or the default drive if none was specified.

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 you find it necessary to modify a file with an extension of .EXE or .HEX, and the exact locations to be modified are known, use the following procedure:

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.

Summary of DEBUG Commands

The following chart is provided for quick reference.

The section called "Format Notation" in Chapter 3 explains the notation used in the format of the following commands.

Command	Purpose	Format
Dump	Displays memory	D [*address*] or D [*range*]
Enter	Changes memory	E *address* [*list*]
Fill	Changes memory blocks	F *range* *list*
Go	Executes with optional breakpoints	G [=*address*] [*address* [*address*. . .]]
Hexarithmetic	Hexadecimal add-subtract	H *value* *value*
Input	Reads/displays input byte	I *portaddress*
Load	Loads file or absolute diskette sectors	L [*address* [*drive* *sector* *sector*]]
Move	Moves memory block	M *range* *address*
Name	Defines files and parameters	N *filespec* [*filespec*. . .]
Output	Sends output byte	O *portaddress* *byte*
Quit	Ends DEBUG program	Q

Command	**Purpose**	**Format**
Register	Displays registers/flags	R [*registername*]
Search	Searches for characters	S *range list*
Trace	Executes and displays	T [=*address*] [*value*]
Unassemble	Unassembles instructions	U [*address*] or U [*range*]
Write	Writes file or absolute diskette sectors	W [*address* [*drive sector sector*]]

APPENDIXES

Contents

APPENDIX A. MESSAGES A-3

APPENDIX B. DOS TECHNICAL INFORMATION B-1
DOS Structure B-1
DOS Initialization B-2
The Command Processor B-3
Replacing the Command Processor . . . B-4
Available DOS Functions B-5
Diskette/File Management Notes B-6
The Disk Transfer Area (DTA) B-7
Error Trapping B-7
General Guidelines B-8
Example of Using DOS Functions B-9

APPENDIX C. DOS DISKETTE ALLOCATION C-1
DOS Diskette Directory C-3
DOS File Allocation Table C-6
How to Use the File Allocation Table C-8

APPENDIX D. DOS INTERRUPTS AND FUNCTION CALLS D-1
Interrupts D-1
Function Calls D-7

APPENDIX E. DOS CONTROL BLOCKS AND WORK AREAS E-1
DOS Memory Map E-1
DOS Program Segment E-2
File Control Block E-7

APPENDIX F. EXE FILE STRUCTURE AND LOADING F-1

APPENDIX A. MESSAGES

The first word of the description of each message 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 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 the diskette was being loaded. 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 disk space freed

CHKDSK. Diskette space marked as allocated was not 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 usually 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 repeated 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 occur again.

A to abort the program.

I to ignore the error condition and continue the program. (This response is not recommended because data is lost when you use it.)

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 ended abnormally 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. Problem resides in object module created from an assembler source program. Debug machine language processor source program; then rerun LINK.

Duplicate filename 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.

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 enter 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 usually occurs when comparing nontext files; it 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 comparison 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, you can:

- Make copies of both files (use COPY command).
- Delete the original files (use ERASE command).
- 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 hyphens (-) and slashes (/).

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 of a second.

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 ended abnormally.

Split the long line into shorter lines; then issue the Replace Text command again.

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 (use 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, the operation is only performed once.

Not ready error writing drive x
Abort, Retry, Ignore?

DOS. See the message **Disk error reading drive** x. In this case, the operation is only performed once.

Out of paper

DOS. Either the printer is out of paper or the printer is not powered ON.

Out of space on list file

LINK. This error usually occurs when there is not enough disk space for the List file.

Out of space on run file

LINK. This error usually occurs 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 the printer 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 ≤ 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 a segment requirement of greater than 64K bytes. The addressing limit is 64K bytes.

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. The names exceeded approximately 50K bytes. Use shorter and/or fewer names.

Terminate 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.

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 eight 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

DISKCOPY. Six attempts were made to read the data from the source diskette. DISKCOPY is unable to continue; therefore, the copy is incomplete.

Unrecoverable verify error on target

DISKCOPY. Six attempts were made to verify the write operation to the target diskette. DISKCOPY is unable to continue; therefore, the copy is incomplete.

Unrecoverable write error on target

DISKCOPY. Six attempts were made to write the data to the target diskette. DISKCOPY is unable to continue; therefore, 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 contain 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 DOS was 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 that 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 8088 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. (Refer to 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’), and 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‘27’ 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’ (terminate address), X‘23’ (CTRL-BREAK handler), X‘24’ (critical error handling) and X‘27’ (terminate but stay resident), as well as a routine to reload the transient portion if needed. (When a program terminates, 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’s 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 terminate 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.

Note: COMMAND.COM should only be replaced by experienced programmers because of the significant amount of function in the DOS command processor.

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 terminate 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' (terminate address), X'23' (CTRL-BREAK handler), X'24' (critical error handling) and if needed X'27' (terminate 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. (Refer to DOS Interrupts and Function Calls in Appendix D.)

Unlike the terminate 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 terminating.

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 correct 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 terminate 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 terminates. 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).

Creating 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.

Loading and Executing 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 terminate program TEST if CTRL-BREAK were pressed).

TEST should:

1. Set the terminate and CTRL-BREAK exit vectors (call X'25') to point to code within itself (the terminate address is where the program to be loaded will return to when it terminates).

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 terminate 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 terminates via INT X‘20’, DOS restores interrupt vectors X‘22’ and X‘23’ from the values in the terminating program's Program Segment Prefix (the values established in step 1) and passes control to the terminate 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 just presented was intentionally simplified by using a memory-image (.COM) program and by omitting discussions of checking the return codes provided by the DOS function calls.
>
> Loading an .EXE file is more complicated due to the file's structure and the need to resolve addresses. Refer to Appendix F for detailed information about the .EXE file structure and loading.

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. (Refer to 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, either:

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, and is written to track 0 sectors 2 and 3 whenever the status of diskette space changes. (See the DOS Memory Map.)

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 terminate. 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 terminate 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. Refer to "Function Calls" in this appendix.

22 Terminate address. The address represented by this interrupt is the address to which control transfers when the program terminates. 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 terminate address prior to creating the segment into which the new program will be loaded. Otherwise, when the second program executes, its termination would cause transfer to its host's termination 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 back slash, 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 termination 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 error on diskette, register AL contains the failing drive number (0 = drive A, etc.); 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 (the first item described below is at the top of the stack), and will contain the following from top to bottom:

IP	DOS registers from issuing INT X'24'
CS	
FLAGS	
AX	User registers at time of original INT X'21' request
BX	
CX	
DX	
SI	
DI	
BP	
DS	
ES	
IP	From the original interrupt X'21' from the user to DOS
CS	
FLAGS	

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 restore your registers from the stack (above). The first and last three words shown should be removed from the stack and discarded. 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 are 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 Terminate 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 terminate. The terminate and CTRL-BREAK exit addresses are restored to the values they had on entry to the terminating 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 terminate 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 points 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 on 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 are 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.

Refer to the section called "DOS Diskette Allocation" in Appendix C 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:DX 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 returns 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 termination 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 terminates.

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) are read from the file address specified by the random record field into the disk transfer address. If end-of-file is reached before all records have been read, AL returns either 01 or 03. A return of 01 indicates end-of-file and the last record is complete. A return of 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 aborted 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
00B1:0000	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’ (terminate), X‘23’ (CTRL-BREAK), X‘24’ (critical error), X‘27’ (terminate 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 by issuing an INT 20, or by issuing an INT 21 with register AH=0. (The instruction INT 20 is the first item in the control block.)

> **Note:** It is the responsibility of all programs to ensure that the CS register contains the segment address of the Program Segment Prefix when terminating 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' (terminate and CTRL-BREAK exit addresses) from the values saved in the Program Segment Prefix of the terminating program. Control is then given to the terminate 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 .EXE programs:

- DS and ES registers are set to point to the Program Segment Prefix. (A diagram of the Program Segment Prefix is provided in this section.)

- CS, IP, SS, and SP registers are set to the values passed by the linker.

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'.

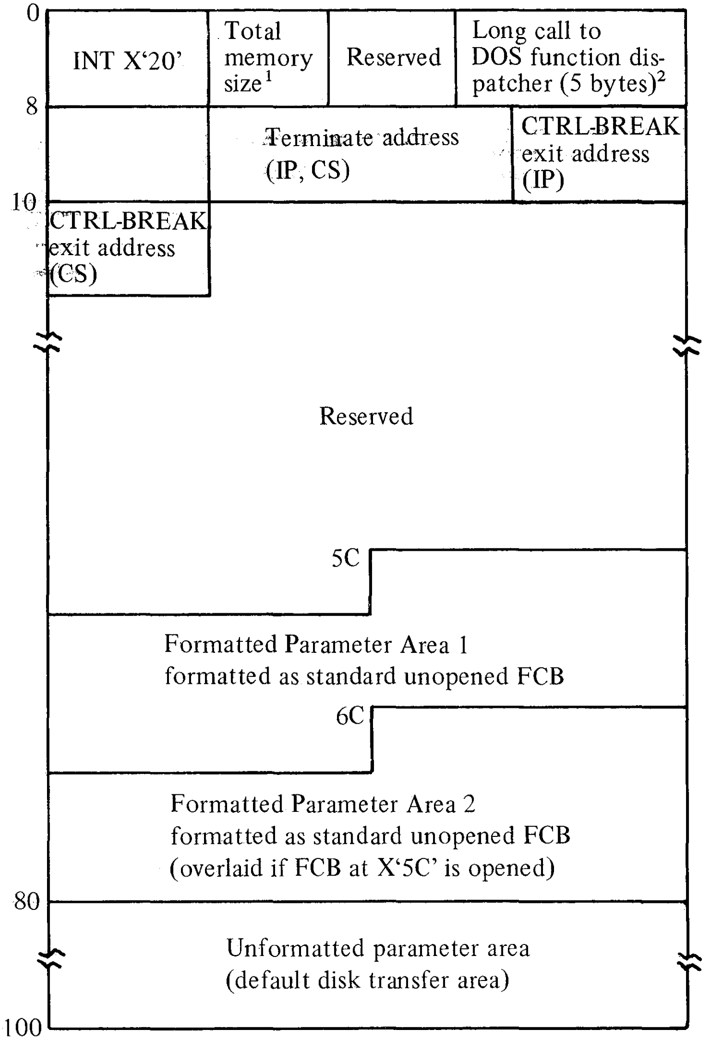
- SP register is set to the end of the program's segment, or the bottom of the transient portion of COMMAND.COM, whichever is lower. The segment size at offset 6 is reduced by X'100' to allow for a stack of that size.

- A word of zeros is placed on the top of the stack.

The Program Segment Prefix (with offsets in hexadecimal) is formatted as follows.

PROGRAM SEGMENT PREFIX

(offsets in hex)



0

INT X‘20’

Total memory size[1]

Reserved

Long call to DOS function dispatcher (5 bytes)[2]

8

Terminate address (IP, CS)

CTRL-BREAK exit address (IP)

10

CTRL-BREAK exit address (CS)

Reserved

5C

Formatted Parameter Area 1 formatted as standard unopened FCB

6C

Formatted Parameter Area 2 formatted as standard unopened FCB (overlaid if FCB at X‘5C’ is opened)

80

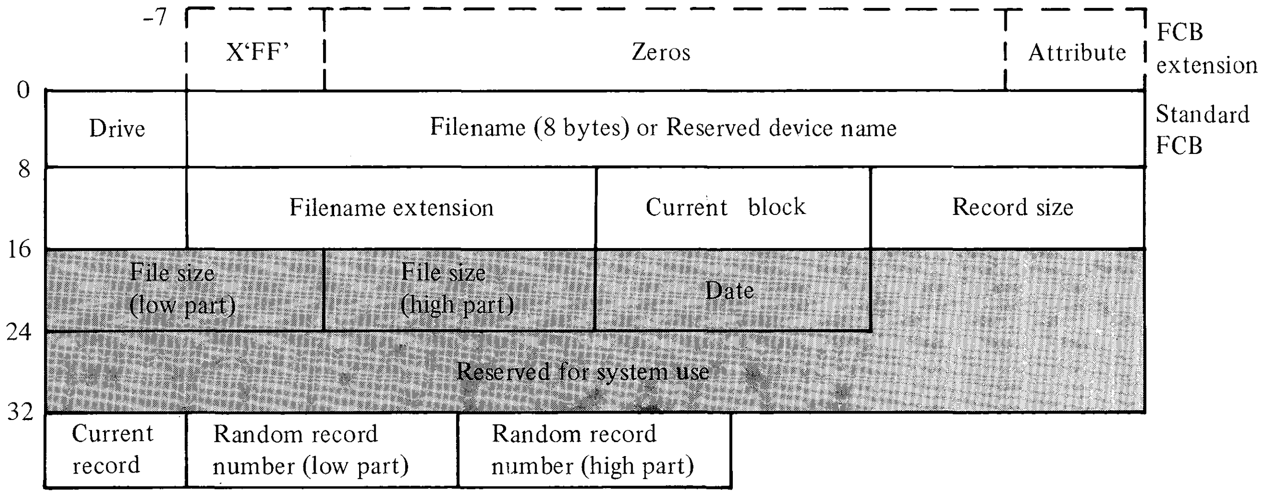
Unformatted parameter area (default disk transfer area)

100

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.

Byte **Function**

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 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 (refer to 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.

APPENDIX F. EXE FILE STRUCTURE AND LOADING

The .EXE files produced by the Linker program consist of two parts:

- control and relocation information
- the load module itself

The control and relocation information, which is described below, is at the beginning of the file in an area known as the *header*. The load module immediately follows the header. The load module begins on a sector boundary and is the memory image of the module constructed by the linker.

The header is formatted as follows:

Hex Offset	**Contents**
00-01	X‘4D’, X‘5A’–This is the LINK program’s *signature* to mark the file as a valid .EXE file.
02-03	Reserved for future use.
04-05	Size of the file in 512-byte increments (*pages*), including the header.
06-07	Number of relocation table items that follow the formatted portion of the header.
08-09	Size of the header in 16-byte increments (*paragraphs*). This is used to locate the beginning of the load module in the file.

Hex Offset	Contents
0A-0B	Reserved for future use.
0C-0D	High/low loader switch. If 0, the load module is to be loaded into high memory. If X‘FFFF’, the load module is to be loaded into low memory.
0E-0F	Offset of stack segment in load module (in segment form).
10-11	Value to be in the SP register when the module is given control.
12-13	Word checksum—sum of all the words in the file, ignoring overflow.
14-15	Value to be in the IP register when the module is given control.
16-17	Offset of code segment within load module (in segment form).
18-19	Offset of the first relocation item within the file.
1A	Reserved for future use.

The relocation table follows the formatted area just described. The relocation table is made up of a variable number of relocation items. The number of items is contained at offset 06-07. The relocation item contains two fields—a 2-byte offset value, followed by a 2-byte segment value. These two fields contain the offset into the load module of a word which requires modification before the module is given control. This process is called *relocation* and is accomplished as follows:

1. A Program Segment Prefix is built following the resident portion of the program that is performing the load operation.

2. The formatted part of the header is read into memory (its size is at offset 08-09).

3. The load module size is determined by subtracting the header size from the file size. Offsets 04-05 and 08-09 can be used for this calculation. Based on the setting of the high/low loader switch, an appropriate segment is determined to load the load module. This segment is called the *start segment*.

4. The load module is read into memory beginning at the start segment.

5. The relocation table items are read into a work area (one or more at a time).

6. Each relocation table item segment value is added to the start segment value. This calculated segment, in conjunction with the relocation item offset value, points to a word in the load module to which is added the start segment value. The result is placed back into the word in the load module.

7. Once all relocation items have been processed, the SS and SP registers are set from the values in the header and the start segment value is added to SS. The ES and DS registers are set to the segment address of the Program Segment Prefix. The start segment value is added to the header CS register value. The result, along with the header IP value, is used to give the module control.

INDEX

Special Characters

. – period 4-7, 4-16
← backspace key 2-14
& – LINK command character 5-16
! – LINK command character 5-16
$$$ – filename extension 4-5
* – asterisk 2-20
* – EDLIN prompt 2-20, 4-4, 4-10
* – global filename character 3-9
- (DEBUG prompt) 6-11
→key 2-16, 2-23
/P option 5-11
/S option 3-46, C-5
% (percent sign) 3-17
? – global filename character 3-9
# – pound sign 4-6
@ character 2-26
= equal sign 3-5
\ back slash 2-13, 2-22
↵ Enter key 2-10

A

A> prompt 2-5, 3-13
abort program B-7
absolute disk read D-5
absolute disk write D-6
absolute diskette sectors B-2
absolute sectors 6-43
absolute segment address 5-25
absolute track/sector, calculate C-4, C-8
AC flag set condition 6-35
access, random B-6
adding hexadecimal values 6-22
address–DEBUG parameter 6-6
address terminate interrupt D-1
address, disk transfer 6-5
Advanced BASIC 2-12
AH register D-7
allocating diskette space B-6, C-6
allocation table address D-15
allocation, diskette C-1
Alt key 2-4
ampersand–LINK command character 5-16, 5-18
analyze
  diskettes 3-38
  the directory 3-19
  the File Allocation Table 3-19
angle brackets 5-18
Append Lines command 4-11, 4-34
architecture, 8088 B-1
ASCII characters 6-12
ASCII codes, extended D-8
ASCII representation 6-5
ASCII values 6-8
Assembler 5-4
asterisk 2-20
  EDLIN prompt 2-20, 4-4, 4-10
  global filename character 3-9

Asynchronous Communications Adapter B-1, D-9
attribute, byte E-10
attribute, file C-3
AUTOEXEC file B-4
AUTOEXEC.BAT file 2-7, 3-15, 3-27
Automatic Program Execution 2-7
Automatic Response File 5-18
  example of 5-19
AUX:–reserved device name 3-8
Auxiliary Asynchronous Communications Adapter B-1, D-9
auxiliary carry flag 6-35
auxiliary input D-9
auxiliary output D-9
available functions, DOS B-5
AX register 6-4, D-4, D-8

B

back slash (\) 2-13, 2-22, D-2
backing up diskettes 1-7
backspace key (←) 2-14
backup diskette 1-7, 3-33
backup file, edit 4-5
BADTRACK 3-29, 3-37, 3-38, C-5
BAK filename extension 4-5, 4-18, 4-26
BASIC Program Editor 2-15
BAT filename extension 3-14
batch file 2-7, 3-14, 3-43, 3-44, 3-47, 3-50
batch file processor B-4
batch processing 3-14
BIOS D-5
BIOS interface module B-1
block number, current E-8
block read, random D-16
block write, random D-16, D-18
blocking/deblocking, data B-1
boot record program 1-3, 1-7, B-1, B-2, C-1
boundary 5-6
  paragraph 5-6
boundary, 16-byte 6-12
boundary, 8-byte 6-12
BP register 6-4, D-4
brackets, square 3-5
Break key 2-10
breakpoint 6-19
buffer, input 2-15
buffered keyboard input D-10
buffers, file D-1
built-in functions B-1
BX register 6-4, D-4
byte–DEBUG parameter 6-6
byte contents 6-15
  display 6-15
  fill 6-18
  replace 6-15
byte, attribute E-10
byte, flag E-10

C

calculate absolute track/sector C-4, C-8
calls, function D-7
carry flag 6-35
chaining file sectors B-6
change date 3-27
change diskettes 3-43
change filenames 3-45
change time 3-47
check keyboard status D-10
checksum methodology B-3
CHKDSK command 3-19, 3-39, 3-50
CL register D-8
class 5-7
clear condition 6-34
close file D-12

codes, error D-3
codes, return B-13
codes, 8088 instruction 6-20
colon 3-5
Color/Graphics Monitor Adapter 3-41
COM filename extension 3-4, B-4
COM programs E-5
comma 3-5
command parameters
  DEBUG 6-6
  DOS 3-6
  EDLIN 4-6
command processor 1-4, 2-5, 2-7, B-3
command processor, replacing the B-4
command processor, resident portion of B-3
command prompt, DEBUG 6-11
command prompt, DOS 2-5
command prompts, LINK 5-8
COMMAND.COM 1-4, 2-4, 3-38, 6-4, B-3, C-4, E-3
  replacing B-4
commands
  DEBUG 6-10
  DOS 3-12
  EDLIN 4-9
commands, end 3-13
commands, summary of
  DEBUG 6-47–6-48
  DOS 3-50–3-51
  EDLIN 4-34
Communications Adapter, Auxiliary Asynchronous B-1, D-9
COMP command 3-21, 3-33, 3-50
comparing diskettes 3-32
comparing files 3-21
Compiler 5-4
COM1:–reserved device name 3-8
CON:–reserved name for console/keyboard 3-8
console I/O, direct D-9
console/keyboard 3-8
console/keyboard routines B-5
control blocks E-1
control keys 2-10, 3-13, 4-9, 6-11
COPY command 1-8, 3-14, 3-23, 3-33, 3-36, 3-50
copying diskettes 1-7, 3-34
copying DOS files 1-7
copying files 3-23
correcting input lines 2-15
create file D-14
creating a .BAT file 3-16
creating a backup diskette 1-7
creating a batch file 3-14
creating a new file 4-19
critical error handler B-3, B-5
critical error handler vector D-3
critical error handling B-3
CS register 6-4, 6-19, 6-21, 6-24, 6-41, 6-44, D-1, D-4, E-3, E-4
Ctrl key 2-4, 2-10
CTRL-BREAK exit address B-4, B-5, D-2
CTRL-BREAK handler B-3, B-5
Ctrl-Break keys 2-10, 2-21, 3-13, 3-15, 3-43, 4-9, 4-16, 4-19, 5-9, 5-17, 6-10
Ctrl-Enter keys 2-11
Ctrl-NumLock keys 2-11, 3-13, 4-9, 6-10
Ctrl-PrtSc keys 2-12, 3-49, 5-12
Ctrl-Z keys 4-28
current block number E-8
current disk D-15
current relative record number E-9
CX register 6-4, 6-5, 6-26, 6-45, D-4
CY flag set condition 6-35

D

d: 3-6
  default 3-6
  parameter 3-6
data area C-1
data blocking/deblocking B-1
date 2-4
  change 3-27
  delimiters 3-28
  enter 2-4, 3-27
  get D-19
  set D-19
DATE command 3-27, 3-50
date file created or updated E-9
deblocking/blocking, data B-1
DEBUG program 6-3
  command parameters 6-6
  commands 6-10
  common information 6-10
  how to end 6-31
  how to start 6-4
  prompt 6-11
  summary of commands 6-47–6-48
default disk transfer address 6-5
default drive 2-5, 3-6
  changing the 2-5
  specifying the 2-5
default prompts 5-9
default segment 6-6
defective tracks 3-38, C-4
Del key 2-16, 2-21
delete file D-13
Delete Lines command 4-12, 4-34
deleting files 3-37
delimiters 3-12, 4-9, 6-10
destination area 6-27
device handler, I/O 1-3
device names, reserved 3-8, 3-13, 3-25
DGROUP 5-15
DI flag clear condition 6-35
DI register 6-4, D-3, D-4
DIR command 3-9, 3-29, 3-39, 3-50
direct console I/O D-9
direction flag 6-35
directory, analyze 3-19
directory 1-6, C-1
directory entries, listing 3-29
directory searches C-3
disk D-11
  current D-15
  reset D-11
  select D-11
Disk BASIC 2-12
disk error handling B-3
disk read, absolute D-5
disk transfer address 6-5, E-4
disk transfer address, set D-15
Disk Transfer Area (DTA) B-7
disk write, absolute D-6
DISKCOMP command 3-32, 3-36, 3-50
DISKCOPY command 1-8, 3-32, 3-34, 3-50
Diskette/File Management B-6
diskettes 1-5
  allocating space B-6, C-6
  allocation C-1
  analyze 3-38
  back up 3-33
  backing up 1-7
  change 3-43
  comparing 3-32
  copying 1-7, 3-34
  creating a backup 1-7
  damaged 1-7
  defective tracks 1-8, 3-38, C-4
  directory C-3
  filenames 3-6
  formatting 1-7
  fragmented 3-35
  handling routines B-5
  initialize 3-38
  new 1-7
  preparing 3-38
  recording format 3-38
  status report 3-19

- display
  - byte contents 6-15
  - flags 6-33
  - instructions 6-40
  - lines 4-22
  - registers 6-33
  - remarks 3-44
- display output D-2, D-8
- displaying memory 6-12
- divide-by-zero B-1
- DN flag set condition 6-35
- DOS 1-3, 2-5
  - available functions B-5
  - command parameters 3-6
  - control blocks E-1
  - copying DOS files 1-7
  - default 5-9
  - diskette allocation C-1
  - diskette directory C-3
  - how to start 2-3
  - Initialization B-2
  - memory map E-1
  - parts of 1-3
  - program segment E-2
  - prompt 2-5
  - structure B-1
  - technical information B-1
  - work areas E-1
- DOS commands 3-4
  - common information 3-12
  - external 3-4
  - internal 3-4
  - summary of 3-50—3-51
  - types of 3-4
- DOS editing keys 2-15, 2-19, 3-13, 4-9, 6-11
  - examples using 2-19—2-29
- drive 3-6
- drive—DEBUG parameter 6-7
- DS register 6-4, 6-5, 6-15, 6-18, 6-27, D-4, E-4
- DSAllocation prompt 5-9, 5-15
- DTA (Disk Transfer Area) B-7
- dummy device 3-8
- dummy parameters 3-16, 3-18
- Dump command 6-12, 6-47
- DX register 6-4, D-4

E

- edit 4-5
  - backup file 4-5
  - existing file 4-5
  - partial file 4-11
- Edit Line command 4-16, 4-34
- editing keys 2-15, 3-13, 4-9, 6-11
- editing template 2-15
- EDLIN 2-15, 2-19, 3-14, 4-3, 4-4
  - prompt 4-4
  - command parameters 4-6
  - commands 4-9
  - common information 4-9
  - how to start 4-4
- EDLIN commands, summary of 4-34
- EDLIN prompt 2-20
- EI flag set condition 6-34
- ellipsis 3-5
- emptying the template 2-27
- End Edit command 4-5, 4-18, 4-33
- ending commands 3-13
- Enter command 6-15, 6-47
- enter date 2-4, 3-27
- Enter key 2-10, 2-15, 3-8, 4-9
- enter time 3-47
- entries, search for D-12
- equal sign (=) 3-5
- ERASE command 3-37, 3-50
- erasing files 3-37
- error codes D-3
- error handler B-7
- error handling B-3
  - critical B-3
  - disk B-3
- error messages A-3
- error trapping B-7
- error, syntax 6-11
- ES register 6-4, 6-5, D-4, E-4
- Esc key 2-13, 2-16, 2-22, 4-16
- exclamation point—LINK command character 5-16
- EXE file structure F-1

EXE filename extension 3-4, 5-11, 6-5, 6-26, 6-45, B-4, E-3
EXE files, load F-1
EXE programs D-7, E-4
execute instructions 6-38
execute program 6-19
executing a .BAT file 3-17
existing file, edit 4-5
.ext 3-7
extended ASCII codes D-8
extensions 3-7
extensions, filename 1-5
  .BAK 4-5, 4-18, 4-26
  .COM 3-4, B-4
  .EXE 3-4, 5-11, 6-5, 6-26, 6-45, B-4, E-3
  .HEX 6-5, 6-26, 6-45
  .MAP 5-12
  .OBJ 5-10
  $$$ 4-5
  length in characters 1-5
external commands 3-4, B-4

F

FAT (see File Allocation Table)
FCB (see File Control Block)
file 1-5
  attribute C-3
  automatic response 5-4
  close D-12
  create D-14
  date created or updated E-9
  delete D-13
  displaying contents of 3-49
  erasing 3-37
  fragmented 3-24
  hidden C-3, D-14, E-10
  how DOS keeps track of 1-6
  input 5-4
  length in characters 1-5
  library 5-4
  listing 5-4
  naming convention 1-5
  object 5-4, 6-3
  object program 3-49
  open D-11
  output 5-4
  per diskette 1-5
  random B-6
  rename D-15
  run 5-4
  sectors B-6
  sequential B-6
  size D-16
  source 4-3
  system E-10
  text 3-49, 4-3
File Allocation Table (FAT) 1-6 B-2, B-6, C-1, C-6
  how to use C-8
file buffers D-1
File Control Block (FCB) 6-28, B-6, E-7
file sectors B-6
  chaining B-6
  mapping B-6
file size E-9
file structure, .EXE F-1
filename characters, global 3-9
filename extensions 1-5, 3-4, 3-7, C-3, E-8
  .BAK 4-5, 4-18, 4-26
  .BAT 3-14
  .COM 3-4, B-4
  .EXE 3-4, 5-11, 6-5, 6-26, 6-45, B-4, E-3
  .HEX 6-5, 6-26, 6-45
  .MAP 5-12
  .OBJ 5-10
  $$$ 4-5
  characters, valid 3-7
  length in characters 1-5
  separators D-18
  terminators D-18
filename, parse D-18
filenames 3-6, 3-8, C-3, E-8
  characters, valid 3-6
  length of 3-6
  renaming 3-45
filespec 3-7

filespec–DEBUG parameter 6-7
Fill command 6-18, 6-47
First Asynchronous Communications Adapter port 3-8
flag byte E-10
flag values 6-32
flags 6-4
flags, display 6-34
FORMAT command 1-3, 1-7, 1-8, 3-38, 3-46, 3-50, C-3
format notation 3-5
formatting diskettes 1-7
fragmented diskettes 3-35
fragmented files 3-24
function calls D-7
functions, available DOS B-5
functions, built-in B-1
F1 key 2-16, 2-23, 2-26
F2 key 2-17, 2-24
F3 key 2-17, 2-21, 2-22, 2-24, 6-24
F4 key 2-17, 2-25
F5 key 2-18, 2-26, 4-17
F6 key 3-8, 3-16, 3-26, 4-28

G

generating line numbers 4-3
get
  date D-19
  time D-19
global filename characters 3-9, 3-13, 3-22, 3-23, 3-29, 3-37, 3-45
  examples using 3-10
global symbols 5-8
Go command 6-19, 6-45, 6-47
group 5-7

H

header F-1
HEX filename extension 6-5, 6-26, 6-45
Hexarithmetic command 6-22, 6-47
hidden file C-3, D-14, E-10
high memory 5-15, 6-5, B-4
high/low loader switch F-3

I

I/O device handler 1-3
IBMBIO.COM 1-3, 3-29, 3-37, 3-38, B-1, B-2, B-3, C-4
  location on diskette C-2
IBMDOS.COM 1-3, 3-29, 3-37, 3-38, B-1, B-2, C-4
  location on diskette C-2
initialization, DOS B-2
initialize diskettes 3-38
input buffer 2-15
Input command 6-23, 6-47
input files 5-4
input, auxiliary D-9
Ins key 2-18, 2-27
Insert Lines command 4-19, 4-34
insert mode 2-22, 2-26, 2-27, 4-19
inserting characters 2-27
inserting lines 4-3
instruction codes, 8088 6-20
Instruction Pointer (IP) 6-4
instruction set, 8088 B-1
instructions 6-38
  display 6-40
  execute 6-38
  unassemble 6-40
  variable length 6-40
INT X‘24’ B-7
INT 21 B-6
interface module, BIOS B-1
internal command processors B-4
internal commands 3-4
interrupt codes 6-20
interrupt flag 6-35
interrupt mechanism, 8088 B-1

interrupt vectors B-2
interrupt X'20' D-1
interrupt X'22' B-4, B-5, D-1
interrupt X'23' B-4, B-5, D-2
interrupt X'24' B-5, D-3
interrupt X'25' D-5
interrupt X'26' D-6
interrupt X'27' B-5, D-6
interrupt, set D-17
interrupts D-1
IP (Instruction Pointer) 6-4
IP register 6-19, 6-32, D-4, E-4
IRET D-2

K

keyboard 3-8
keyboard input D-8
keyboard input, buffered D-10
keyboard status, check D-10
keys, control 2-10, 3-13, 4-9, 6-11
keys, DOS editing 2-15, 3-13, 4-9, 6-11
keywords 3-5

L

Libraries prompt 5-8, 5-12
line–EDLIN parameter 4-6
Line Editor Program 2-15, 2-19, 4-3
line numbers 4-3
Line Numbers prompt 5-9, 5-13
linefeed D-2
lines, renumber 4-12, 4-19
LINK 5-3
  example session 5-20
  how to start 5-17
LINK command prompts 5-8
linker files 5-4
Linker program 5-3
Linker, example session 5-20
list–DEBUG parameter 6-7
List File prompt 5-8, 5-11
List Lines command 4-22, 4-34
listing directory entries 3-9, 3-29
Load command 6-24, 6-45, 6-47
Load Low prompt 5-6, 5-9, 5-14
load module 5-13, 5-23
load module memory map 5-24
loading .EXE files F-1
locate next sector C-6, C-8
logical record numbers D-6
logical record size E-8
low memory 5-11, 5-15
LPT1:–reserved name for printer 3-8

M

MAP filename extension 5-12
mapping file sectors B-6
memory 5-11
  high 5-15, 6-5, B-4
  low 5-11, 5-15
memory management routine B-6
memory map E-1
memory status report 3-19
memory, display 6-12
memory, loading files into 4-4
messages A-3
messages, LINK 5-26
MODE command 3-41, 3-51
Monochrome Display Printer Adapter 3-41
MOV instruction C-8
Move command 6-27, 6-47
multi-drive system 2-8

N

n–EDLIN parameter 4-7

NA flag clear condition 6-35
Name command 6-25, 6-28, 6-47
NC flag set condition 6-35
new diskette 1-7
NG flag set condition 6-35
notation, format 3-5
NUL:—reserved device name 3-8
numbers, line 4-3, 4-12
NV flag clear condition 6-35
NZ flag clear condition 6-35

O

OBJ filename extension 5-10
object files 6-3
Object Modules prompt 5-8, 5-10
object program files 3-49
open file D-11
operation, suspend system 2-11, 3-43
optional remarks, PAUSE command 3-43
Output command 6-30, 6-47
output files 5-4
output routines B-5
output, auxiliary D-9
output, display D-2, D-8
OV flag set condition 6-35
overflow flag 6-35

P

paragraph boundary 5-6
parameters 3-6, 4-6, 6-10
  DEBUG 6-6
  DOS 3-6
  dummy 3-16, 3-18
  EDLIN 4-6
  save area E-4
  testing with different 6-19
parity flag 6-35
parse filename D-18
partial file, edit 4-11
PASCAL 5-12, 5-14
PAUSE command 3-43, 3-51
PC register 6-32
PE flag set condition 6-35
percent sign (%) 3-17
period (.) 4-7, 4-16
PL flag clear condition 6-35
PO flag clear condition 6-35
portaddress—DEBUG parameter 6-7
pound sign (#) 4-6
prepare diskettes 3-38
print displayed output 2-12
print screen output 2-13
print string D-10
printer 2-12, 3-8, 3-41
printer output D-9
printer output routines B-5
PRN:—reserved name for printer 3-8
program execution, stop 6-19
program segment D-17, E-2
  create new D-17
Program Segment Prefix 6-5, B-3, B-5, B-7, E-4, E-6
program terminate D-8
public symbols 5-11, 5-22
Publics prompt 5-8, 5-13
punctuation 3-5

Q

question mark 3-5
Quit command 6-31, 6-47
Quit Edit command 4-26, 4-34
quotation marks 6-8

R

random access B-6
random block read B-6, D-17
random block write B-6, D-18

random read D-16
random record field, set D-17
random write D-16
range–DEBUG parameter 6-7–6-8
Read-Only Memory (ROM) B-1
read, random D-16
read, random block D-17
read, sequential D-13
record number, relative E-9
record numbers, logical D-6
record size, logical E-8
recording format, diskette 3-38
Register command 6-32, 6-48
registername–DEBUG parameter 6-8
registernames, valid 6-32
registers, display 6-33
relative record number E-9
relative record number, current E-9
relative sector number 6-8, C-3
relative zero 5-22
relocatable loader 5-4
relocation F-2
REM command 3-44, 3-51
remarks, display 3-44
remarks, PAUSE command 3-43
RENAME command 3-45, 3-51, 6-46
rename file D-15
renumber lines 4-12, 4-19
replace byte contents 6-15
Replace Text command 4-27, 4-34
replacing characters 2-27
replacing the command processor B-4
reserved device names 3-8, 3-13, 3-25
reset, disk D-11
reset, system 2-4, B-2
resident portion of command processor B-3
return codes B-13
ROM (Read-Only Memory) B-1
ROM BIOS routine D-9
routines B-1
  console/keyboard B-5
  device B-1
  diskette handling B-5
  keyboard input B-5
  memory management B-6
  output B-5
  printer output B-5
  ROM BIOS D-9
  time function B-5
Run File prompt 5-8, 5-11

S

save area, parameter E-4
screen 3-8
screen display 2-12
  restart 2-12
  suspend 2-12
screen output, print 2-13
Search command 6-37, 6-48
search for entries D-12
Search Text command 4-30, 4-34
sector–DEBUG parameter 6-8
sector boundary C-1
sector number, relative 6-8, C-3
sector, locate next C-6, C-8
sector, starting C-3
sectors 1-6, 3-35, 6-8
  absolute 6-43
sectors, file B-6
segments 5-10
segment 5-6
SEGMENT command 5-14
segment registers 6-4, 6-41
segment, create new program D-17
segment, default 6-6
segment, start F-3
segments 5-7

select disk D-11
semicolon delimiter 3-12
separators, filename D-18
sequential read D-13
sequential write D-14
set
  date D-19
  interrupt D-17
  random record field D-17
  time D-19
set condition 6-34
set disk transfer address D-15
Shift key-PrtSc keys 2-13
SI register 6-4, D-4
sign flag 6-35
single-drive system 2-8, 3-33, 3-35, D-11
size, file D-16, E-9
slashes 3-5
source area 6-26
source drive 3-13
source files 4-3
SP (Stack Pointer) 6-4
SP register E-4, E-5
space allocation B-6, C-1
space delimiter 3-12
special characters 3-9
specifying a drive 3-6
square brackets 3-5
SS register 6-4, E-4
stack allocation statement 5-14
Stack Pointer (SP) 6-4
Stack Size prompt 5-9, 5-13
stack space B-2
stack, user D-4
start segment F-3
starting DEBUG 6-4
starting DOS 2-3
  computer power off 2-3
  computer power on 2-4
starting EDLIN 4-4
starting LINK 5-17
starting sector C-3
status report 3-19
stop program execution 6-19
string–DEBUG parameter 6-8–6-9
string–EDLIN parameter 4-8, 4-9
structure, DOS B-1
suspend screen output 2-12
suspend system operation 2-11, 3-43
switch, high/low loader F-3
symbols 5-8
  global 5-8
  public 5-11
syntax error 6-11
SYS command 3-46, 3-51
system devices 3-8
system file E-10
system files, transfer 3-46
system prompt 2-8, 3-13, B-4
system reset 2-4, B-2

T

target drive 3-13
technical information, DOS B-1
template 2-15
temporary file, VM.TMP 5-5
terminate address B-5
terminate address interrupt D-1
terminate but stay resident B-3
terminate commands 3-13
terminate program D-8
terminate program interrupt D-1
terminators, filename D-18
text files 3-49, 4-3
thrashing B-9
time 3-47
  change 3-47
  enter 3-47
  get D-19
  set D-19
TIME command 3-47, 3-51
time function routines B-5
Trace command 6-38, 6-45, 6-48
track/sector, calculate absolute C-4
tracks, defective 3-38, C-4
transfer address, disk E-4

transfer system files 3-46
transient portion of command processor B-3
TYPE command 3-49, 3-51
types of DOS commands 3-4

U

Unassemble command 6-40, 6-48
unassemble instructions 6-40
unprintable characters 6-12
UP flag clear condition 6-35
user stack D-4

V

value 6-9
variable length instructions 6-40
VM.TMP temporary file 5-5, 5-10, 5-11

W

work areas E-1
Write command 6-43, 6-48
Write Lines command 4-33, 4-34
write, random D-16
write, random block D-18
write, sequential D-14

Z

zero flag 6-35
ZR flag set condition 6-35

IBM

The Personal Computer
Software Library

Product Comment Form

Disk Operating System 6172220

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in anyway it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Comments:

If you wish a reply, provide your name and address in this space.

Name ______________________________

Address ____________________________

City ______________ State ______________

Zip Code __________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 123 BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

Tape

Please do not staple

Tape

IBM

The Personal Computer
Software Library

Product Comment Form

Disk Operating System 6172220

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in anyway it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Comments:

If you wish a reply, provide your name and address in this space.

Name ______________________________

Address ______________________________

City ______________ State ______________

Zip Code ___________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 123 BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

Please do not staple

Tape

The Personal Computer
Software Library

Product Comment Form

Disk Operating System 6172220

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in anyway it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Comments:

If you wish a reply, provide your name and address in this space.

Name ____
Address ____
City ____________________ State ____________________
Zip Code _______________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 123 BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

Please do not staple

Tape

IBM

The Personal Computer
Software Library

Product Comment Form

Disk Operating System 6172220

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in anyway it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Comments:

If you wish a reply, provide your name and address in this space.

Name ______________________________

Address ______________________________

City ______________ State ______________

Zip Code ______________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 123 BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

Please do not staple

Tape

Continued from inside front cover

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

IBM does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, IBM warrants the diskette(s) or cassettes on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

LIMITATIONS OF REMEDIES

IBM's entire liability and your exclusive remedy shall be:

1. the replacement of any diskette or cassette not meeting IBM's "Limited Warranty" and which is returned to IBM or an authorized IBM PERSONAL COMPUTER dealer with a copy of your receipt, or
2. if IBM or the dealer is unable to deliver a replacement diskette or cassette which is free of defects in materials or workmanship, you may terminate this Agreement by returning the program and your money will be refunded.

IN NO EVENT WILL IBM BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH PROGRAM EVEN IF IBM OR AN AUTHORIZED IBM PERSONAL COMPUTER DEALER HAS BEEN ADVISED OF THE POSSIBLITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

GENERAL

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

This Agreement will be governed by the laws of the State of Florida.

Should you have any questions concerning this Agreement, you may contact IBM by writing to IBM Personal Computer, Sales and Service, P.O. Box 1328–W, Boca Raton, Florida 33432.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.