Microsoft
MS™-DOS
Disk Operating
System

MS-DOS OPERATING SYSTEM DOCUMENTATION

Documentation for the MS-DOS operating system is provided in two manuals, each of which is described below:

The MS-DOS User's Guide

This manual gives an overview of the MS-DOS operating system, describes the user interface, the file system, the command structure, and each of the available commands. It also contains chapters on:

```
EDLIN.COM  - The MS-DOS line editor
DEBUG.COM  - The MS-DOS debugger
FILCOM.COM - The MS-DOS file comparison program
```

The Utility Software Package Manual

This manual provides descriptions of the following software:

```
MASM.EXE - The MACRO-86 macro assembler
LINK.EXE - The MS-LINK linker
CREF.EXE - The MS-CREF cross-reference utility
LIB.EXE  - The MS-LIB library manager
```

MS™-DOS user's guide

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. The software described in this document is furnished under a license agreement or non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the MS-DOS Disk Operating System on cassette tape, disk, or any other medium for any purpose other than the purchaser's personal use.

LIMITED WARRANTY

MICROSOFT CORPORATION shall have no liability or responsibility to purchaser or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this product. This product will be exchanged within twelve months from date of purchase if defective in manufacture, labeling or packaging, but except for such replacement the sale or subsequent use of this program is without warranty or liability.

THE ABOVE IS A LIMITED WARRANTY AND THE ONLY WARRANTY MADE BY MICROSOFT CORPORATION. ANY AND ALL WARRANTIES FOR MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY EXCLUDED.

To report software bugs or errors in the documentation, please complete and return the Problem Report at the back of this manual.

Document No. 8410-110-01
Catalog No. 1406
Part No. 14F07A

Microsoft

Welcome to the Microsoft family of products.

The Microsoft Corporation is the recognized leader in microcomputer software. The Microsoft BASIC interpreter, in its several versions, is the standard high-level programming language for microcomputers. The Microsoft Corporation provides other software of consistently high-quality that sets the standard for software quality.

In addition to the MS-DOS Disk Operating System and the Microsoft BASIC interpreter, Microsoft sells other full-feature language compilers, operating systems, and utility software. For all this software, Microsoft strives to offer a family of software products that both look alike from one product to the next, and can be used together for effective program development.

For more information about other Microsoft products, contact:

Microsoft Corporation
The Microsoft Building
10700 Northup Way
C-97200
Bellevue, WA 98004
(206) 828-8080

Package Contents

1 disk with the following files:

CHKDSK.COM
COMMAND.COM
CREF.EXE
DEBUG.COM
EDLIN.COM
EXE2BIN.EXE
FILCOM.COM
FORMAT.COM
IO.SYS (hidden file)
LIB.EXE
LINK.EXE
MASM.EXE
MSDOS.SYS (hidden file)
SYS.COM

2 Manuals:

The MS-DOS Disk Operating System User's Manual
The Microsoft Utility Software Package Manual

System Requirements

The MS-DOS Operating System requires an 8086 or 8088 microcomputer system. The operating system itself runs in and requires 32K bytes of memory.

CONTENTS

CHAPTER 1

INTRODUCTION

The MS-DOS disk operating system is one of Microsoft's family of operating systems for 8086 and 8088 microprocessors. It provides a simple but powerful interface between the user and a computer system's resources. Most all Microsoft languages are available under MS-DOS, including the BASIC Interpreter, the BASIC Compiler, MS-Pascal, and MS-FORTRAN. In addition, the 8-bit versions of Microsoft's languages are upward compatible with the 16-bit versions. Thus, application programs written in 8-bit Microsoft languages can be run under MS-DOS with little or no modification.

1.1 FEATURES AND BENEFITS OF MS-DOS

The following features and benefits make MS-DOS the operating system of choice for 8088 and 8086 microcomputers:

Easy Conversion from 8080 to 8086

MS-DOS allows as much transportability of 8-bit machine language software as is reasonably possible. For instance, MS-DOS emulates system calls to the 8-bit CP/M operating system. Therefore, by simply running assembly language source code through the Intel conversion program, almost all 8080 programs created for the CP/M operating system can be made to work without modification in the MS-DOS environment. In most cases, converting programs from CP/M-80 to MS-DOS is easier than converting to other 16-bit operating systems.

Device Independent I/O

MS-DOS simplifies I/O to different peripheral devices by assigning a reserved filename to each device. These names are built-in to MS-DOS and are detected by the MS-DOS file system. Thus, for example, programs designed only for disk file I/O can have their input come from the terminal keyboard or their output sent to the printer.

Advanced Error Recovery Procedures

MS-DOS does not necessarily require rebooting when disk errors occur. If a disk error occurs at any time during any program, MS-DOS retries the operation three times. If the operation cannot be completed successfully, MS-DOS returns an error message, then waits for the user to enter a response. Thus, the user can attempt to recover from the error rather than reboot the operating system.

Complete Program Relocatability

The architecture of the 8086 CPU limits each segment of memory to 64K-bytes and requires intersegment references to be fixed for a given load address. MS-DOS works around this limitation through its special executable object module format. During program development, the Microsoft linker can combine object modules created with any of Microsoft's BASIC, Pascal, or FORTRAN compilers or by Microsoft's macro assembler. These modules can be combined to create an executable module requiring any number of segments.

Powerful, Flexible File Characteristics

In MS-DOS, there is no practical limit on file or disk size. MS-DOS uses 4-byte XENIX Operating System compatible logical pointers for a disk capacity of up to 1 gigabyte. (Microsoft's XENIX is a licensed version of the UNIX system. XENIX is available for a variety of 16-bit microcomputer systems through a license agreement from Microsoft.)

MS-DOS remembers the exact physical end-of-file marker. Thus, should one open a file with a logical record length greater than the physical record length, MS-DOS remembers exactly where the file ends to the byte, rather than rounded to 128 bytes.

Written entirely in 8086 Assembly Language

Because it is written entirely in 8086 assembly language, MS-DOS provides significant speed improvements over operating systems that are largely translated from their 8-bit counterparts.

Fast, Efficient File Structure

MS-DOS employs a highly efficient disk structure which eliminates the need for "extents," minimizes access to the directory track, and provides for duplicate directory information and verifications after writes.

No Need to Log in Disks

As long as no file is currently being written, disks can be swapped without logging in. One benefit of this feature is that the MS-DOS debugger, DEBUG.COM can be used without reloading for different programs on different disks.

No Physical File/Disk Size Limitation

Unlike operating systems that are limited to 8-megabytes per disk, MS-DOS does not require breaking a 24-megabyte hard disk into three separate logical drives.

No Overhead for Non-128-Byte Physical Sectors

Since MS-DOS does its own blocking and deblocking of disk sectors, there is no reason to worry about different physical sector sizes when writing the low level routines for a particular computer system.

Time and Date Stamps

When a file is modified, MS-DOS automatically records the time and date of the modification.

100% IBM Compatible

International Business Machines Corporation has chosen MS-DOS (called IBM Personal Computer DOS) to be the preferred operating system for the IBM Personal Computer. IBM has already announced Microsoft BASIC, Pascal, and FORTRAN along with other accounting, financial planning, and word processing software that runs under MS-DOS.

1.2 PROVIDED SOFTWARE

The software provided with the MS-DOS operating system is described below:

CHKDSK.COM

CHKDSK.COM is a command used to check and verify the contents of a disk. It is further described in Chapter 3, "Commands."

COMMAND.COM

COMMAND.COM is the command interpreter used to interface between the user and the underlying operating system. It allows the user to perform file management functions such as rename and delete, as well as to load and execute programs. COMMAND.COM is further described in Chapter 2, "System Structure."

CREF.EXE

CREF.EXE is the Microsoft MS-CREF cross-reference utility used to create a cross-reference listing from an assembly source listing. CREF.EXE is further described in the Utility Software Package Manual.

DEBUG.COM

DEBUG.COM is a debugger program used to provide a controlled testing environment for executable object files. DEBUG.COM is further described in Chapter 5, "DEBUG."

EDLIN.COM

EDLIN.COM is the MS-DOS line editor. Intraline editing is performed using the special editing keys that are also available at the MS-DOS command level. EDLIN.COM is further described in Chapter 4, "EDLIN."

EXE2BIN.COM

EXE2BIN.COM is used to convert .EXE files to .COM files. In general, only assembly language programs that have been specially formulated may undergo such conversions. EXE2BIN.COM is further described in Chapter 3, "Commands."

FORMAT.COM

FORMAT.COM is used to format disks so that they can be used with MS-DOS. FORMAT.COM is described in Chapter 3, "Commands."

FILCOM.COM

FILCOM is a file comparison program used to check for differences between files. Either text or binary files may be compared. FILCOM is further described in Chapter 6, "FILCOM."

IO.SYS

IO.SYS is the lowest level of the MS-DOS operating system, interfacing to all I/O devices. It is an MS-DOS "hidden file" and does not show up when a directory command is executed. IO.SYS is automatically loaded into memory when your system is booted up. See Chapter 2, "System Structure" for more information.

LIB.EXE

LIB.EXE is the Microsoft MS-LIB library manager used to create, maintain, and manipulate libraries of object files. LIB.EXE is further described in the Utility Software Package Manual.

LINK.EXE

LINK.EXE is the Microsoft MS-LINK linker used to link object files and object libraries to create executable .COM and .EXE files. LINK.EXE is further described in the Utility Software Package Manual.

MASM.EXE

MASM.EXE is Microsoft's relocatable macro assembler for 8086 and 8088 microprocessors, MACRO-86. MASM.EXE is further described in the Utility Software Package Manual.

MSDOS.SYS

MSDOS.SYS is the heart of the MS-DOS operating system, where most management of system resources takes place. MSDOS.SYS is intimately tied to COMMAND.COM and IO.SYS. Note that MSDOS.SYS is an MS-DOS "hidden file" and does not show up when a directory command is executed. MSDOS.SYS is automatically loaded into memory when your system is booted up. For further information, see Chapter 2, "System Structure."

SYS.COM

SYS.COM is used to transfer MSDOS.SYS and IO.SYS from a system disk to a formatted disk that does not contain the MS-DOS operating system on it. It is further described in Chapter 3, "Commands."

1.3 SYSTEM START-UP

To start-up your system, follow your manufacturer's instructions. Next, insert your system disk in drive A:. At this point, your MS-DOS will be booted up and loaded from disk into program memory. A banner then appears containing the MS-DOS version number. Next, COMMAND.COM is loaded into memory and a banner containing appears for it to. Then you are prompted to set the date and time with the DATE and TIME commands.

For example, you might type:

```
    Current time is 00:00:00.00
    Enter new time:10:30
    Current date is 1-1-80
    Enter new date:3-25-82
```

Finally, the MS-DOS prompt appears with the letter of the driver and a colon:

```
    A:_
```

The cursor is indicated in this manual with an underline character. At this point you are at the MS-DOS command level and are under the supervision of COMMAND.COM.

The first thing that you should do is create a back-up of your system disk. This is done by first inserting a blank disk into drive B:. Next type:

```
    FORMAT  B:/S
```

FORMAT formats the disk in drive B: so that it can be used with MS-DOS. The /S causes hidden system files to be copied to the newly formatted disk after it has been formatted. When FORMAT is done, you should then type:

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

This command copies all files on the system disk to the new disk. At this point, you should remove the original system disk and store it in a safe place. Use the copy of the system disk from now on.

1.4 SYNTAX NOTATION

The following notation is used throughout this manual in descriptions of command and statement syntax:

[ ] Square brackets indicate that the enclosed entry is optional.

< > Angle brackets indicate user entered data. When the angle brackets enclose lower case text, the user must type in an entry defined by the text; for example, <filename>. When the angle brackets enclose upper case text, the user must press the key named by the text; for example, <RETURN>.

{ } Braces indicate that the user has a choice between two or more entries. At least one of the entries enclosed in braces must be chosen.

... Ellipses indicate that an entry may be repeated as many times as needed.

CAPS Capital letters indicate portions of statements or commands that must be entered exactly as shown. When capital letters appear within angle brackets, they indicate typing of a control character such as <CONTROL-C> or <RETURN>.

All other punctuation, such as commas, colons, slash marks, and equal signs, must be entered exactly as shown.

CHAPTER 2

SYSTEM STRUCTURE

This chapter gives a structural overview of the MS-DOS operating system, describing:

1. System resources

2. The file system

3. The user interface

4. Command types

By acquiring an understanding of each of these subjects, you will gain a sound understanding of the structure of the MS-DOS operating system.

2.1 SYSTEM RESOURCES

Each time your computer is turned on, it normally will "boot up" the operating system by automatically loading MS-DOS from disk into memory. The area of memory in which the operating system is loaded is referred to as system memory.

MS-DOS, itself, consists of three files:

```
COMMAND.COM
MSDOS.SYS
IO.SYS
```

These three files combine to form an operating system that controls all system resources. Note that MSDOS.SYS and IO.SYS are "hidden" files that are not displayed when a directory command is executed. The relationship between these three files and system resources is shown in Figure 2.1.

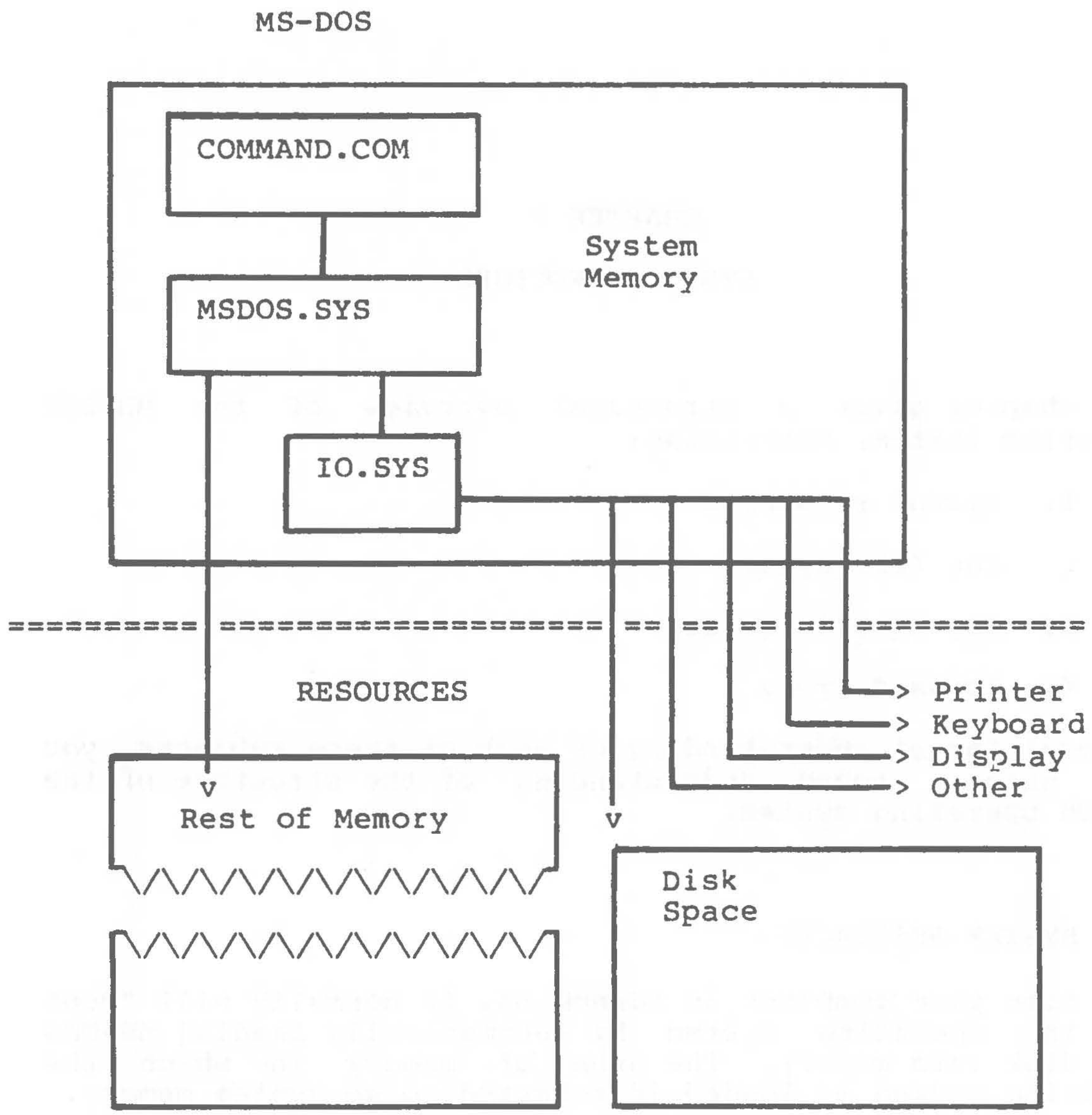


Figure 2.1 MS-DOS and Its Resources

System resources include peripheral devices such as terminals, printers, and serial lines. However, a system's most important system resources are its disk space and its memory--these are described below.

2.1.1 Disk Space

In MS-DOS, disk space is divided into four parts:

The Reserved Sectors
: This region contains information that is used each time MS-DOS is booted up. This information is system dependent, but typically will include a simple bootstrap loader.

The Directory
: The directory contains information about each file on a given disk, including the file's complete filename, its size, and its time and date of last modification.

The File Allocation Table
: The file allocation table (FAT) contains location information for the data making up each file on a given disk. Note that MS-DOS does not require a file's contents to reside in physically contiguous disk sectors.

Files
: The great majority of disk space is reserved for the contents of files. An individual file does not necessarily reside in contiguous sectors on disk, and may be "scattered" in memory to decrease waste of disk space.

2.1.2 Memory

Besides controlling a system's disk space and its other devices, MS-DOS must also control main memory. This means that MS-DOS must be capable of loading files into memory either as data files or as files that are to be executed. The actual loading of files is performed by IO.SYS, the lowest level of the MS-DOS operating system. Loading of executable files is supervised by COMMAND.COM. For most well-designed programs, control is returned to MS-DOS after either normal or abnormal termination of a program.

Note that part of COMMAND.COM may be overlaid to make room for a particularly large executable file. After execution of such a file, MS-DOS automatically loads the overlaid part of COMMAND.COM back into system memory, and normal execution of COMMAND.COM resumes. If the overlaid part of COMMAND.COM is not available on disk because the disk on which it resides has been removed, the following message appears:

```
Insert DOS disk in default drive
and strike any key when ready
```

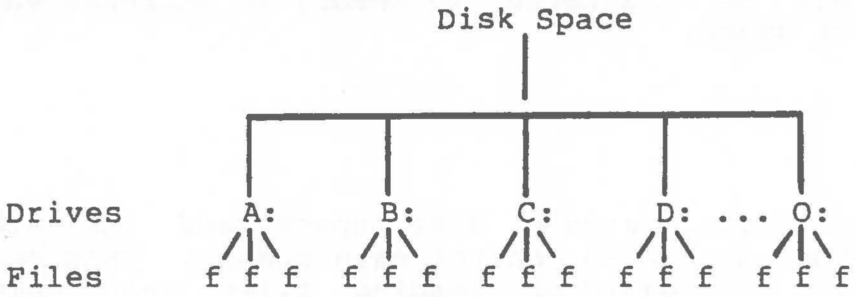
Also, if an incorrect version of COMMAND.COM is found, then the a similar message appears:

```
Invalid COMMAND.COM
Insert DOS disk in default drive
and strike any key when ready
```

2.2 FILE SYSTEM

The preceding discussion of system resources discussed many of the internal aspects of the operating system. A file system, on the other hand, can be thought of as the external organization of system resources. It provides a way of talking about files and devices. Note that MS-DOS supports "device independent I/O", which means that the distinction between files and devices is an internal distinction, but not an external one. Therefore, the user can treat files and devices alike, and can refer to either with "filenames."

Note, however, that disk space is special, since it is divided into drives; the disk space in a drive and on a particular disk is further divided into files, as shown below:



Disks are named so that up to 15 can be referred to. They are named with the letters A through O, where each letter is followed by a colon (:). This colon separates the name of the disk from the names of individual files on the disk. This letter-colon combination is called a drive designation.

2.2.1 Naming Conventions

MS-DOS supports a three-part name for files (and devices) called a file specification. A full file specification contains a drive designation, a filename, and a filename extension.

The format of a file specification is:

```
[<d>:]<filename>[.<ext>]
```

The parts of a file specification are described below:

<d>: is the drive designation as described in the preceding section.

<filename> is an internal name consisting of from 1 to 8 characters. Internally, all filenames are exactly 8 characters: this means that filenames with seven or fewer characters have all remaining characters padded with spaces. MS-DOS performs this padding automatically. Legal characters in filenames are:

```
A-Z    0-9    $    &    #    @    !

%      '      (    )    -    <    >

\      ^      {    }    ~    |    `
```

Any of the above characters is legal in any position in the filename. Note that lowercase characters are converted to uppercase wherever they occur in a file specification. This means that a filename such as "FiLe.ExT" is converted to the filename "FILE.EXT."

.<ext> is the filename extension consisting of three or fewer characters. Padding of spaces occurs as described above for filenames. Legal characters are the same as for filenames. All characters are legal in any position in the extension, and all lowercase characters are converted to uppercase characters, just as for filenames. Filenames with no extensions can also be specified by typing only a period after the filename. It is usually only necessary to use this form when a given command requires a default extension.

2.2.2 Wild Card Characters

There are two wild card characters that can be used in file specifications: the asterisk (*) and the question mark (?). By using these characters, a short hand notation is created for specifying multiple files. This notation is particularly useful when file specifications are required as parameters for commands. MS-DOS makes full use of this capability for commands such as the directory, delete, and copy commands: DIR, DEL, and COPY.

The two wild card characters are described below:

? When COMMAND sees a question mark (?), it matches any <u>single</u> character found in that position in a filename.

For example, examine the following command:

```
DEL AB?DE.EXT
```

This directory command deletes all files whose names begin with AB, end with DE, and have the filename extension .EXT. For example, MS-DOS might delete the following files:

```
ABCDE.EXT
ABODE.EXT
ABIDE.EXT
```

* When COMMAND.COM sees an asterisk (*) in a filename parameter, it matches all characters found in those positions in any filename in the appropriate directory. The asterisk (*) is a shorthand for a series of question marks (?). So, for example, the following are equivalent:

```
DEL *.*
DEL ???????.???
```

However, the asterisk is typed more easily.

The command DEL *.* deletes all files stored on the disk in the default drive. regardless of filename or extension.

Here are some other examples, with their equivalents in question mark characters:

```
DEL FILE.*     DEL FILE.???
(Delete all variations of FILE
 regardless of extension)

TYPE *.EXT     TYPE ????????.EXT
```

```
DEL *.EXT      DEL ????????.EXT
(Delete all files with the
 filename extension .EXT)

DEL ABC*.E*    DEL ABC?????.E??
(Delete all files whose names
 begin with ABC and that have
 a filename extension that
 begins with .E)
```

2.2.3 Device Filenames

Certain 3-letter filenames are reserved for the names of devices. These names are listed below:

AUX

Used when referring to input from or output to an auxiliary device.

CON

Used when referring to either keyboard input or to output to the terminal screen.

LST or PRN

Used when referring to the line printer.

NUL

Used when you do not want to create a particular file, but the syntax of a command requires an input or output filename.

Even if given device designations or extensions, these filenames remain associated with the devices listed above. Thus, A:CON.LST still refers to the terminal console and is not the name of a disk file. This device naming scheme permits treating devices as if they were files, and is a consequence of MS-DOS's device independent I/O.

2.3 USER INTERFACE

MS-DOS acts as an interface between the user and a computer system's resources, with communication from the user normally occurring through keyboard input. This keyboard input is the raw data that is used to edit the MS-DOS command line. When editing is complete, the command line is passed on to COMMAND.COM where it is scanned for command names and parameters. Thus, the user interface consists of two levels of processing: command line editing and command interpretation. These levels are discussed in the next two sections.

2.3.1 Command Line Editing

MS-DOS offers a variety of functions that operate on the command line buffer. These functions make command line editing a simple and efficient task, very much in contrast to the nuisance it can be in other operating systems. The command line buffer is intimately related to another buffer called the "template," which is used in many of MS-DOS's special editing functions.

The model for command line input is as follows:

1. When text is entered from the keyboard, it is held in the command line until the <RETURN> key is pressed.

2. Pressing <RETURN> causes the contents of the command line to be sent to COMMAND.COM for processing.

3. Pressing <RETURN> also copies the command line to the template.

Thus, the template always contains the last entered command line.

The command line is altered by entering one the following kinds of input:

Alphanumerics
Punctuation
Special editing functions
Control character functions

Alphanumeric and punctuation characters are entered into the command line as they are typed. Later, COMMAND.COM will convert all lowercase letters to uppercase.

The special editing and control character functions greatly increase MS-DOS's ease of use. All of the MS-DOS commands, from DEBUG to FILCOM, can make use of these functions wherever input is required from a terminal. These functions are always resident in the operating system.

2.3.1.1 Special Editing Functions - Because they depart from the "normal" way in which most operating systems handle command input, the special editing functions deserve particular emphasis. They relieve the user of repeatedly typing in the same sequences of keys because the last command line entered is automatically placed in the template and "remembered."

Therefore, by using the template and executing the special editing functions, you receive the following advantages:

1. A command line can be instantly repeated in two key-strokes.
2. An erroneous command line can be edited and retried, without reentering the entire command line.
3. A command line similar to a preceding command line can be edited and executed with a minimum of typing.

The relationship between the command line and the template is shown in Figure 2.2.

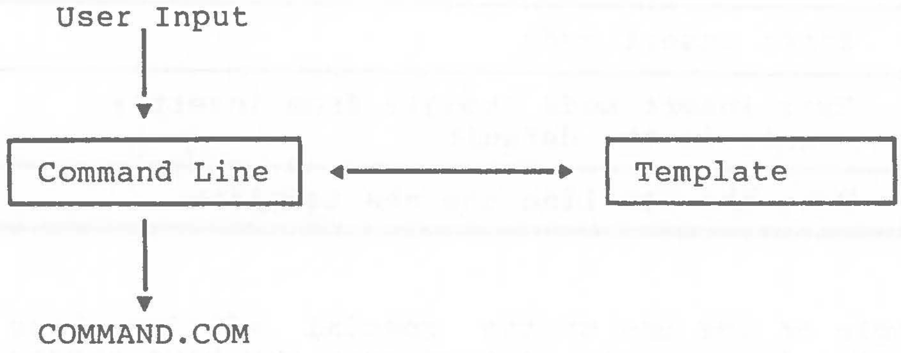


Figure 2.2 The Command Line and the Template

Table 2.1 contains a complete list of the special editing commands. Each command is more fully described in Chapter 4, "EDLIN," where these special editing commands become a subset of the commands available within the MS-DOS editor, EDLIN, and are called the intraline editing commands.

Note that the special editing commands may be assigned to the keys that make the best use of a specific terminal keyboard. Therefore, each command is identified by a functional name rather than by a specific key code. For an application on a specific terminal, the codes are configured for specific terminal keys.

Table 2.2 Special Editing Functions

Key	Editing Function
<C1>	Copy one character from the template to the command line
<CM>	Copy all characters up to the character specified from the template to the command line
<CT>	Copy all remaining characters in the template to the command line
<S1>	Skip over (do not copy) a character in the template
<SM>	Skip over (do not copy) the characters in the template up to the character specified
<QI>	Void the current input; leave the template unchanged
<INS>	Enter insert mode
<REP>	Exit insert mode (toggle from insert); this is the default
<NT>	Make the new line the new template

As an example of the use of the special editing keys and command entry in general, pretend that you have entered the following command:

```
    DIR PROG.COM<RETURN>
```

This command displays contents of the file PROG.COM on the terminal screen. It also has the useful side effect of saving the command line in the template. To repeat the command, all you have to do is type two keys: <CT> and <RETURN>.

The repeated command is displayed on the screen, as you type, as shown below:

```
<CT>DIR PROG.COM<RETURN>
```

Note that pressing the <CT> key causes the contents of the template to be copied to the command line buffer; pressing <RETURN> causes the command line to be processed by COMMAND.COM.

Now pretend that you want to display the contents of the file PROG.ASM. To do this we will make use of the template, and type

```
<CM>C
```

Entering <CM>C copies characters from the template to the command line buffer up to the character "C":

```
DIR PROG._
```

Note that the underline is your cursor. Now type:

```
.ASM
```

The result is:

```
DIR PROG.ASM_
```

The desired command line is now in the command line buffer. To send this command line on to COMMAND.COM, the command interpreter, simply enter <RETURN>.

The template now contains the following command line:

```
DIR PROG.ASM
```

Now assume that we want to execute the following command:

```
TYPE PROG.ASM
```

To do this, we type:

```
TYPE<INS> <CT><RETURN>
```

Note that normal alphanumeric characters are entered directly into the command line buffer, automatically replacing corresponding characters in the template. This automatic replacement is turned off when the <INS> key is typed. Thus, the characters "TYPE" replace the characters "DIR " in the template. To insert a space between "TYPE" and "PROG.ASM," we first typed <INS> and then a space. Finally, to copy the rest of the template to the command line, we typed <CT>, followed by a <RETURN>.

When <RETURN> is typed, the entire command line is copied to the template, in this case:

```
     TYPE PROG.ASM
```

If we had misspelled "TYPE" as "BYTE", a command error would occur. Still, we could save the mistyped command line by creating a new template with the <NT> key:

```
     BYTE PROG.ASM<NT>
```

We can then edit this erroneous command line by typing:

```
     T<Cl>P<CT>
```

The <Cl> key copies a single character from the template to the command line buffer. The resulting command line then is what we want:

```
     TYPE PROG.ASM
```

As an alternative, we could have used the same template containing BYTE *.ASM, and used the <Sl> and <INS> commands to achieve the same result:

```
     <Sl><Sl><Cl><INS>YP<CT>
```

To illustrate how the command line buffer is affected as you type, examine the keys typed on the left and their affect on the contents of the command line buffer, shown on the right:

```
     <Sl>         _                 {Skips over 1st template char}
     <Sl>         _                 {Skips over 2nd template char}
     <Cl>         T_                {Copies 3rd template char}
     <INS>YP      TYP_              {Inserts two characters}
     <CT>         TYPE PROG.ASM_    {Copies rest of template}
```

Note that <Sl>, like <SM>, does not affect the command line buffer; rather, it effects the template by deleting the first character in the template. Similaraly, <SM> deletes characters in the template up to a given character (this character is the next one typed).

As you can see from the above examples, these special editing function keys can add greatly to your effectiveness at the keyboard. The next section describes control character functions that complement the above functions.

2.3.1.2 Control Character Functions - While commands are being entered, MS-DOS recognizes seven control character functions. These control characters and the functions associated with them are shown in Table 2.2.

Table 2.2 Control Character Functions

Control Character	Function
<CONTROL-N>	Cancel echoing of output to line printer.
<CONTROL-C>	Abort current command.
<CONTROL-H>	Remove last character from command line, and erase character from terminal screen.
<CONTROL-J>	Insert physical end-of-line, but do not empty command line. Use Linefeed to extend the current logical line beyond the physical limits of one terminal line.
<CONTROL-P>	Echo terminal output to the line printer.
<CONTROL-S>	Suspend display of output to terminal screen. Press any key to resume.
<CONTROL-X>	Cancel the current line, empty the command line, and then output a back slash (\), carriage return, and line feed. The template used by the Special Editing commands is not affected.

2.3.2 Command Interpretation

The MS-DOS user interface permits editing of command lines with the special editing and control character functions described in the preceding section. Once a command line has been edited, it is sent to COMMAND.COM for processing. COMMAND.COM is the hub of the operating system, acting as the interface between the lower levels of the operating system and user input. It is in COMMAND.COM that the commands that are entered on the command line are interpreted.

Commands themselves are of one of two types: either internal or external. Internal commands are all resident in memory as part of COMMAND.COM; they are loaded into memory when the operating system is booted up. External commands, on the other hand, are loaded into memory from disk only when needed. External commands reside in disk files that have a name with either a .COM or .EXE extension. Note that COMMAND.COM itself is an external command. A picture of the command interface is shown in Figure 2.3.

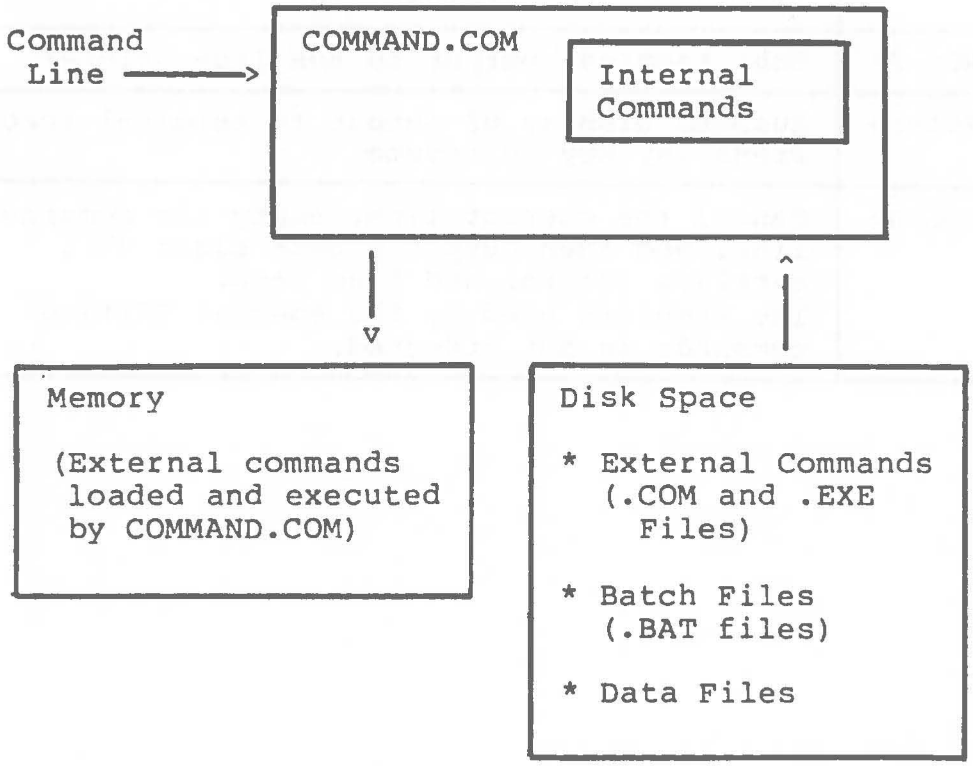

Command Line
COMMAND.COM
Internal Commands
Memory
(External commands loaded and executed by COMMAND.COM)
Disk Space
* External Commands (.COM and .EXE Files)
* Batch Files (.BAT files)
* Data Files

Figure 2.3 MS-DOS Command Structure

Command interpretation begins when COMMAND.COM scans the command line for the name of a legal command. If it is an internal command, it is executed immediately; if it is a batch file, commands are executed indirectly from a .BAT file; if it is an external .COM or .EXE command, the appropriate file is loaded from disk into memory where it is executed. (The batch facility and .BAT commands are discussed later in this chapter.)

COMMAND.COM provides MS-DOS' characteristic colon prompt (:) in the form of a drive designation letter and a colon (:).

For example:

```
     A:_
```

The cursor is the focus of any editing actions that you perform. The cursor is indicated by an underline in this manual; the symbol itself is implementation dependent.

At system start-up, the default prompt is always A:. After start-up, the user may specify the default (that is, the currently selected drive). To select a new drive, simply enter the designation letter followed by a colon:

```
     A:_                (prompt; default drive)
     A:B:<RETURN>       (user enters new drive designation)
     B:_                 (new prompt; new default drive)
```

Note that COMMAND's main goals are to identify commands typed at the command line and then to execute them. All commands consist of a command name followed by optional parameters. When parameters are present, they must be separated from the command name and from each other. Spaces, tabs, and commas are the only legal separators.

For example:

```
     COPY OLDFILE.REL,NEWFILE.REL

     RENAME THISFILE THATFILE
```

COMMAND.COM is able to execute several different types of commands: these types are described later in this chapter.

2.4 COMMAND TYPES

COMMAND.COM allows you to execute four different types of commands:

1. Internal commands such as DIR, REN, TYPE and DEL.
2. .COM commands such as CHKDSK.COM, DEBUG.COM, and EDLIN.COM.
3. .EXE commands such as LINK.EXE, MASM.EXE, and CREF.EXE.
4. .BAT command files that contain multiple instances of the above commands.

The internal commands and the simplest, most commonly used external commands are described in Chapter 3, "Commands." You should study all of these commands carefully.

Use of the MS-DOS batch facility is discussed in the next section of this chapter. The batch facility allows you to indirectly execute a set of commands contained in a file. This facility allows you to tailor commands for a particular purpose without a great deal of programming effort.

Separate chapters are also provided on three utility programs:

EDLIN	- The MS-DOS line editor
DEBUG	- The MS-DOS debugger
FILCOM	- The MS-DOS file comparison program

Finally, most of the .EXE files provided with the MS-DOS operating system are described in the <u>Utility Software Package Manual</u>. The programs described in this manual make up a powerful and complete assembly language development system.

This software includes:

MASM.EXE	- The MACRO-86 macro assembler for 8088 and 8086 microprocessors
LINK.EXE	- The MS-LINK linker
CREF.EXE	- The MS-CREF assembler cross-reference utility
LIB.EXE	- The MS-LIB library manager facility

2.4.1 Internal Commands

Internal commands are incorporated in COMMAND.COM and are always available when COMMAND.COM is resident in memory. Unlike the external commands, discussed in the next section, these commands need not be available on disk when they are executed. Note that most internal commands are simple and easy to use. This is in contrast to some of the external commands which are larger and more complex.

The internal commands are listed below. Each is described thoroughly in the next chapter.

COPY
DATE
DEL
DIR
REM
REN
PAUSE
TIME
TYPE

2.4.2 External Commands

Any file with the filename extension .COM or .EXE is considered valid as an external command. Such commands are executed by entering the name of the file less its .COM or .EXE extension. Programs that you create with most languages will be .EXE files.
External commands include:

ASM86
CHKDSK
COMMAND
CREF
DEBUG
EDLIN
EXE2BIN
FORMAT
FILCOM
LINK
LIB
SYS

Note that .EXE files created with the MACRO-86 assembler can be converted to .COM files with the command EXE2BIN.EXE. The format of a .COM file is special, so .EXE files cannot be arbitrarily converted. Note also that all .COM commands execute in less than 64K of memory; .EXE files, on the other hand, may require more than 64K of memory to execute.

2.4.3 Batch Commands

The MS-DOS batch facility allows files containing commands to be submitted for processing internally by MS-DOS. "Batches" of commands in such files are processed as if they were typed at a terminal. Each batch file must be named with the .BAT extension, and is submitted for execution by entering its filename less that extension. Optional parameters may be given as well. Therefore, the invocation syntax is as follows:

```
<filespec>[<parameters>]
```

By creating a .BAT file with prototype commands containing positional parameters, parameters may be passed to the .BAT file when it is executed. You may specify up to 10 positional parameters, named %0 through %9.

The parameters are substituted in their order on the invocation line for corresponding occurrences in the batch file. If the dummy parameter %0 is used, the batch facility substitutes the name of the batch command itself for parameter %0. Thus, the batch facility permits creation of batch commands that can be used on more than just one set of files, and that can be used to reexecute themselves.

For example, a batch file might look like this when viewed from within EDLIN, the MS-DOS line editor:

```
1: REM This is file NEWDISK.BAT
2: REM   (the .BAT extension must be given)
3: PAUSE  Insert disk in B:
4: FORMAT B:/S
5: DIR B:
6: CHKDSK B:
```

To execute this .BAT file, simply enter the filename without the .BAT extension:

```
NEWDISK
```

The result is the same as if each of the lines in the .BAT file were entered at the terminal as individual commands.

To pass parameters to the .BAT file, the user must create a .BAT file containing prototype commands with dummy entries.

For example:

```
1: REM This is A:ASMFILE.BAT
2: REM START BATCH FILE
3: COPY %1.ASM %2.ASM
4: MASM %2,%2,%2;
5: TYPE %2.PRN
6: TYPE %0.BAT
```

Assume that this file exists as A:ASMFILE.BAT.

To execute this .BAT file and pass parameters, enter:

```
A:ASMFILE A:MYPROG B:MYPROG
```

The result is the same as if you had entered each of the following commands at your terminal:

```
REM This is A:ASMFILE.BAT
REM START BATCH FILE
COPY A:MYPROG.ASM B:MYPROG.ASM
MASM B:MYPROG,B:MYPROG,B:MYPROG;
TYPE B:MYPROG.PRN
TYPE A:ASMFILE.BAT
```

When you boot up your system, COMMAND.COM searches for the file A:AUTOEXEC.BAT. If a file with that name exists on disk, then the batch facility is automatically invoked to execute the commands contained in AUTOEXEC.BAT. In such a case, execution of the TIME and DATE commands at start-up is bypassed. If COMMAND.COM does not find AUTOEXEC.BAT, then the normal MS-DOS prompt is displayed instead.

Two MS-DOS commands are available expressly for their use in batch files: REM and PAUSE. REM permits the inclusion of remarks and comments in batch files; PAUSE prompts the user with an optional message and permits either continuing or aborting execution of a batch file at a given point. REM and PAUSE are further described in Chapter 3, "Commands."

CHAPTER 3

COMMANDS

NOTE

Users of single-drive systems should refer to Appendix A for the additional procedures required when executing many of the following commands.

The following notation is used in the descriptions in this chapter.

filespec Refers to an optional drive designation followed by a filename followed by a period and an optional three letter filename extension. For example:

B:ABODE.BAS	(refers to a disk file on disk B:)
FILE.BAS	(refers to a disk file on the default disk)
CON	(refers to the user's terminal console)
CON.BAS	(same as above)

filename Refers to any valid name for a disk file, including an optional extension. A filename parameter does not refer to a device or to a disk drive designation alone.

d: Refers to a disk drive designation

All of the commands described in this chapter are listed below:

CHKDSK Scan the directory of the default or designated drive and check for consistency.

COPY Copy file(s) specified.

DATE Display and set date.

DEL Delete files specified. ERASE is a synonym for this command.

DIR List requested directory entries.

EXE2BIN Convert .EXE file to a .COM file.

FORMAT Format a disk to receive MS-DOS files.

PAUSE Pause for input in a batch file.

REM Display a comment in a batch file.

REN Rename first file as second file. RENAME is a synonym for this command.

SYS Transfer MS-DOS system files from drive A: to the drive specified.

TIME Display and set time.

TYPE Display the contents of file specified.

NAME TYPE

CHKDSK External

SYNTAX

```
CHKDSK [d:]
```

FUNCTION

Scan the directory of the default or designated drive and check it for consistency.

COMMENTS

CHKDSK should be run occasionally on each disk to verify the integrity of the directory structure. If any errors are found, the appropriate error message is displayed and corrective action is attempted.

After the disk has been checked, CHKDSK displays error messages, if any, and then a status report.

A sample status report follows:

```
160256   bytes total disk space
  8192   bytes in 2 hidden files
 30720   bytes in 8 user files
121344   bytes available on disk

 65536   bytes total memory
 53152   bytes free
```

If an error is detected, CHKDSK returns one of the following error messages:

Allocation error for file <filename>

The named file had a data block allocated to it that did not exist (that is, a data block number larger than the largest possible block number). CHKDSK truncates the file short of the bad block.

Disk not initialized

No directory or file allocation table was found. If files exist on the disk, and the disk has been physically harmed, it may still be possible to transfer files from this disk to recover data.

Directory error-file: <filename>

No valid data blocks are allocated to the named file. CHKDSK deletes the file.

Files cross-linked: <filename> and <filename>
The same data block is allocated to both files. No corrective action is taken. To correct the problem, first use the COPY command to make copies of both files; then, delete the originals. Review each file for validity and edit as necessary.

File size error for file <filename>
The size of the file in a directory is different from its actual size. The size in the directory is automatically adjusted to indicate its actual size on the disk. (The amount of useful data may be less than the size shown because the last data block may not be used fully.)

XXXXXX bytes of disk space freed
Disk space shown as allocated was not actually allocated and has been freed.

NAME
COPY

TYPE
Internal

SYNTAX
COPY filespec [filespec]

FUNCTION
Copy the first filespec to the second.

COMMENTS

If the second filespec parameter is not given, the copy is on the default drive and has the same name as the original (first filespec parameter). If the first filespec is on the default drive and the second filespec is not given, the COPY is aborted. (Copying files to themselves is not allowed.) MS-DOS returns the error message:

```
File cannot be copied onto itself
0 File(s) copied
```

The second parameter may take three forms. If the second parameter is a drive designation (d:) only, the original file is copied with the same name to the designated drive. If the second parameter is a filename only, the original file is copied to a file with the name specified on the default drive. If the second parameter is a full filespec, the original file is copied to a file with the name specified on the designated drive.

The COPY command also allows file concatenation while copying. Concatenation is invoked by simply listing any number of files as parameters to COPY, separated by "+".

For example,

```
COPY  A.XYZ + B.COM+B:C.TXT BIGFILE.CRP
```

The above command concatenates the contents of A.XYZ, B.COM, and B:C.TXT and places them in the file on the default drive called BIGFILE.CRP.

The concatenation operation is normally carried out in text (or ASCII) mode, meaning a <CONTROL-Z> (1A hex) in the file is interpreted

The concatenation operation is normally carried out in text (or ASCII) mode, meaning a <CONTROL-Z> (1A hex) in the file is interpreted as the end-of-file mark. To combine binary files, this interpretation of the end-of-file may be overridden with the /B switch, which forces the command to use the physical end-of-file as the end of file (that is, the file length seen in the DIR command).

For example,

```
COPY/B A.COM + B.COM
```

Also, in the above example, no resulting file name was given. In this case, COPY seeks to the end of A.COM and appends B.COM to it, leaving the result named A.COM.

ASCII and binary files may be arbitrarily combined by using /B on binary files and /A on ASCII files. A switch (/A or /B) takes effect on the file it is placed after and applies to all subsequent files until another switch is found.

A /A or /B switch on the destination file determines whether or not a <CONTROL-Z> is placed at the end of the file. (Source files read while /A is in effect have <CONTROL-Z> stripped off. If /A is in effect when the file is written, a single <CONTROL-Z> will be put back.) Thus, an additional <CONTROL-Z> would be appended with a command such as:

```
COPY  A.ASM/B B.ASM/A
```

This occurs because the /B on the first file prevents the <CONTROL-Z> from being stripped, and the /A on the second puts one on. The primary practical application may be the reverse, where a <CONTROL-Z> is stripped from the file.

For example:

```
COPY PROG.COM/B + ERRS.TXT/A NEWPROG.COM/B
```

It is assumed here that ERRS.TXT was generated by an editor, but is actually considered constant data (error messages) by the program it is being appended to. Since the result is a

Even when <u>not</u> concatenating files, the /A and /B switches are still processed. When not concatenating, the copy command defaults to binary copy. By using the /A switch, the result file may be truncated at the first end-of-file mark:

```
COPY A.TXT/A B.TXT
```

B.TXT may be shorter than A.TXT if A.TXT contained an embedded <CONTROL-Z>. B.TXT will have exactly one <CONTROL-Z>, the last character of the file.

Concatenation with ambiguous file names is allowed, and the COPY command normally "does what you want". To combine several files specified with an ambiguous name into one file, use a command like:

```
COPY *.LST COMBIN.PRN
```

All files matching *.LST are combined into one file called COMBIN.PRN. Another type of task is performing several individual concatenations:

```
COPY *.LST + *.REF *.PRN
```

In this example, for each file found matching *.LST, that file is combined with the corresponding .REF file, with the result given the same name but with the extension .PRN. Thus, FILE1.LST will be combined with FILE1.REF to form FILE1.PRN, then XYZ.LST with XYZ.REF to form XYZ.PRN, and so on. The following COPY command combines all files matching *.LST, then all files matching *.REF, into one file call COMBIN.PRN:

```
COPY *.LST + *.REF COMBIN.PRN
```

It is easy to enter a concatenation COPY command where one of the source files is the same as the destination, yet this often cannot be detected. For example, the following command is an error if ALL.LST already exists:

```
COPY *.LST ALL.LST
COPY *.LST ALL.LST
```

This is not detected, however, until it is ALL.LST's turn to be appended. At this point it could already have been destroyed.

COPY handles this problem like this: as each input file is found, its name is compared with the destination. If they are the same, that one input file is skipped, and the message "Content of destination lost before copy" is printed. Further concatenation proceeds normally. This allows "summing" files, with a command like

```
COPY ALL.LST + *.LST
```

This command appends all *.LST files, except ALL.LST itself, to ALL.LST. The error message is suppressed in this case, since this is produced by a true physical append to ALL.LST.

NAME
DATE

TYPE
Internal

SYNTAX

```
DATE [<mm>-<dd>-<yy>]
```

FUNCTION

Display and set the date.

COMMENTS

If entered without a parameter, DATE returns with the message:

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

Press <RETURN> if you do not want to change the date shown.

Optionally, the date may be given as a parameter to the DATE command as in:

```
DATE 3-9-81
```

In this case, no message appears.

The new date must be entered using numerals only: letters are not permitted. The allowable parameters are:

```
<mm> = 1-12
<dd> = 1-31
<yy> = 80-99 or 1980-2099
```

The date, month, and year entries may be separated by hyphens (-) or slashes (/). MS-DOS is programmed to change months and years correctly, whether the month has 31, 30, 29, or 28 days. (Note that MS-DOS handles leap years, too.)

If the parameters or separators are not legal, MS-DOS returns the message:

```
Invalid date
Enter new date:_
```

DATE then waits for entry of a legal date.

NAME
DEL

TYPE
Internal

SYNTAX

```
DEL   filespec
```

FUNCTION

Delete all the files with the filespec specified.

COMMENTS

If the filename is *.*, the prompt "Are you sure?" appears. If a "Y" or "y" is typed as a response, then all files are deleted as requested. ERASE is a synonym for this command.

NAME

DIR

TYPE

Internal

SYNTAX

```
DIR [filespec][/P][/W]
```

FUNCTION

List the files in a directory

COMMENTS

If no parameter is present (DIR), all directory entries on the default drive are listed. If only the drive specification is present (DIR d:), all entries on the disk in the specified drive are listed. If only filename is present (DIR filename) with no extension, then all files with the filename specified on the disk in the default drive are listed. If a full file specification is present (DIR d:filename.ext), all files with the filename specified on the disk in the drive specified are listed. In all cases, files are listed with their size in bytes and the time and date of their last modification.

The wild card characters question mark (?) and asterisk (*) may be used in the filename parameter. Refer to Section 2.2.2, "Filenames," for examples of the use of the wild card characters. Note that for the convenience of the user, the following invocations of the DIR command are equivalent:

COMMAND	EQUIVALENT
DIR	DIR *.*
DIR FILE	DIR FILE.*
DIR .EXT	DIR *.EXT
DIR .	DIR *.

Two switches may be given with DIR. The /P switch selects Page Mode. With /P, display of the directory pauses after the screen is filled. To resume display of output, type any key.

The /W switch selects Wide Display. With /W, only file names are displayed without other file information. Files are displayed five per line.

NAME
EXE2BIN

TYPE
External

SYNTAX

```
EXE2BIN filespec [d:][filename][.ext|
```

FUNCTION
Convert files from .EXE format to binary format

COMMENTS

The first parameter is the input file; if no extension is given, it defaults to .EXE. The second parameter is the output file. If no drive is given, the drive of the input file is used; if no filename is given, the filename of the input file is used; if no extension is given, .BIN is used.

The input file must be in valid .EXE format produced by the linker. The "resident", or actual code and data part of the file, must be less than 64K. There must be no STACK segment. Two kinds of conversion are possible depending on the specified initial CS:IP:

1. If CS:IP is not specified, a pure binary conversion is assumed. If segment fix-ups are necessary, the following prompt appears:

   ```
   Fix-up needed - base segment (hex):
   ```

   By typing a legal hexadecimal number and then <RETURN>, execution will continue.

2. If CS:IP is specified as 100H, then it is assumed the file is to be run as a .COM file ORGed at 100H, and the first 100H of the file is to be deleted. No segment fix-ups are allowed, as .COM files must be segment relocatable.

If CS:IP does not meet one of these criteria or meets the .COM file criterion, but has segment fix-ups, the following error message is displayed:

```
File cannot be converted
```

Note that to produce standard .COM files with the MACRO-86 assembler, one must both ORG the file at 100H and specify the first location as the start address (this is done in the END statement).

For example:

```
            ORG     100H
    START:
             .
             .
             .
            END     START
```

NAME

FORMAT

TYPE

External

SYNTAX

```
FORMAT d:[/S]
```

FUNCTION

Format the disk in the drive designated to accept MS-DOS files.

COMMENTS

Initialize the directory and file allocation tables. The reserved sectors are copied onto track 0, sector 1. (This occurs whether or not the /S switch is given.)

If the /S switch is present, FORMAT copies operating system files from the disk in the default drive to the newly formatted disk. The files copied are copied in the following order:

IO.SYS
MSDOS.SYS
COMMAND.COM

NAME	TYPE
PAUSE	Internal

SYNTAX

```
PAUSE [comment]
```

FUNCTION

Suspend execution of the batch file.

COMMENTS

During the execution of a batch file, you may need to change disks or to perform some other action between the execution of batch commands. The PAUSE command exists for just such purposes. PAUSE suspends execution until you type any key, except <CONTROL-C>.

When COMMAND encounters PAUSE, it prints:

```
Strike a key when ready . . .
```

Pressing any key except <CONTROL-C> resumes execution of the batch file. If you type <CONTROL-C>, another prompt is displayed:

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

If you type "Y" in response to this prompt, execution of the remainder of the batch command file is aborted and control returns to the operating system command level. Therefore, PAUSE can be used to break a batch file into pieces, allowing you to end the batch command file at an intermediate point.

The optional comment may be entered on the same line as PAUSE. You may also want to prompt the user of the batch file with some meaningful message when the batch file has paused. For example, you may want to change disks in one of the drives. An optional prompt message may be given in such cases. The comment prompt is displayed before the "Strike a key" message.

NAME
REM

TYPE
Internal

SYNTAX
REM [comment]

FUNCTION
Display remark entered on same line as REM when encountered during execution of batch file.

COMMENTS
The REM command has no other effect. The only delimiters for the comment are any one of the three legal delimiters to start the comment (blank space, tab, comma).

NAME
REN

TYPE
Internal

SYNTAX

```
REN filespec filename
```

FUNCTION

Change the name of the first parameter (filespec) to the second parameter (filename).

COMMENTS

The first parameter (filespec) must be given a drive designation if the file disk resides in a drive other than the currently logged (default) drive. Any drive designation for the second parameter (filename) is ignored. The file will remain on the disk where it currently resides.

The wildcard characters, question mark (?) and asterisk (*), may be used in either parameter. All files matching the first filespec are renamed. If wildcard characters appear in the second name, corresponding character positions are not changed.

For example, the following command changes the names of all files with the .LST extension to similar names with the .PRN extension:

```
REN *.LST *.PRN
```

Another example causes the file ABODE on drive B: to be renamed ADOBE:

```
REN B:ABODE ?D?B?
```

The file remains on drive B:.

An attempt to rename a file to a name already present in the directory will result in the error message "Duplicate file name or file not found."

Note that RENAME is a synonym for the REN command.

NAME
SYS

TYPE
External

SYNTAX

```
SYS d:
```

FUNCTION

Transfer the MS-DOS system files from the disk in the default drive to the drive specified by d:.

COMMENTS

SYS is normally used to update a system or to place the system on a formatted disk that contains no files. An entry for d: is required.

The files transferred are copied in the following order:

IO.SYS
MSDOS.SYS

Note that COMMAND.COM is <u>not</u> transferred and that IO.SYS and MSDOS.SYS are both hidden files that do not appear when the DIR command is executed.

NAME
TIME

TYPE
Internal

SYNTAX

```
TIME [<hh>[:<mm>[:<ss>]]]
```

FUNCTION
Display and set the time.

COMMENTS
If the TIME command is entered without any parameters, then the following message is displayed:

```
Current time is <hh>:<mm>:<ss>.<cc>
Enter new time:_
```

Simply type <RETURN> if you do not want to change the time shown. Optionally, a new time may be given as a parameter to the TIME command as in:

```
TIME 8:20:00
```

The new time must be entered using numerals only: letters are not allowed. The allowable parameters are:

```
<hh> = 00-24
<mm> = 00-59
<ss> = 00-59
```

The hour, minute, and second entries must be separate by colons.

MS-DOS uses whatever time is entered as the new time as long as the parameters and separators are legal. If the parameters or separators are not legal, MS-DOS returns the message:

```
Invalid time
Enter new time:_
```

MS-DOS then waits for entry of a legal time.

NAME

TYPE

TYPE

Internal

SYNTAX

TYPE filespec

FUNCTION

Display the contents of the file on the console screen.

COMMENTS

Use this command to examine a file without modifying it. Use DIR to find the name of a file and EDLIN to alter the contents of a file. The only formatting performed by TYPE is that tabs are expanded to spaces consistent with tab stops every eighth column. Note that display of binary files causes control characters to be sent to your computer, including bells, formfeeds, and escape sequences.

CHAPTER 4

EDLIN

EDLIN is a text editor used to edit files that are divided into lines. Each line may be up to 255 characters, the last character of each being the end of line character, the carriage return. Line numbers are not actually present in saved text, but when a file is displayed, lines are numbered dynamically. When a file is created or edited, line numbers begin at 1 and are incremented by one through the end of the file. When new lines are inserted between existing lines, all line numbers following the inserted text are automatically incremented by the number of lines inserted. When lines are deleted between existing lines, all line numbers following the deleted text are decremented automatically by the number of lines deleted. Consequently, lines numbers always run from 1 through n (the last number).

4.1 INVOCATION

To invoke EDLIN, enter:

```
     EDLIN <filespec>
```

If the file specified exists, EDLIN loads the file into memory. If the whole file is loaded, EDLIN returns the message "End of input file" and an asterisk (*) prompt. If the file is "larger than memory", then EDLIN fills 3/4 or available memory with the first part of the file and then returns the asterisk (*) prompt, but not the "End of input file" message. (This is just like the Append command with no parameter. See Section 4.3, "Interline Commands," for more information on Append.)

You may then edit the existing file. When you want to edit the part of a file that is not in memory, you must first write out to disk some of the file that is in memory, and then append lines into memory.

These commands are:

```
[<n>]W
```

where <n> is the number of lines to be written out; and

```
[<n>]A
```

where <n> is the number of lines to be appended.

When the editing session ends, the file is saved on the same drive where it was found by typing:

```
E
```

If the file specified does not exist, EDLIN creates the file and returns the message NEW FILE. and then displays the asterisk (*) prompt, indicating that the editing session may begin.

IMPORTANT

> When creating a new file, be sure to specify on which drive the file should be saved. The command to end the editing session and save the file does not allow parameters. Therefore, if the drive is not designated during EDLIN invocation, the file is saved on the default drive.

EDLIN commands belong to two types: intraline and interline. Intraline commands perform editing functions within a single line. The commands used to perform intraline editing are the control character functions and the special editing commands discussed in Chapter 2. The special editing functions are described in more detail in the following section than they were in Chapter 2. Note, however, that these are the same commands that are used at the MS-DOS command level. The only difference between them is that the EDLIN commands operate on the line currently being edited, rather than the MS-DOS command line.

4.2 INTRALINE COMMANDS

Intraline commands include the special editing functions and the control character functions: only the special editing functions are discussed here. See Section 2.3.1.2 for more information on the control character functions.

The special editing commands all may be assigned to the keys that make the best use of a specific terminal keyboard. Therefore, each command is identified by a functional name rather than by a specific key, and each is configurable to a particular keyboard key code. A code has been given to each command for ease of reference during the examples which demonstrate the function. (For an application on a specific terminal, the codes should be replaced by the names of the specific terminal keys.) Table 4.1 summarizes the commands, codes, and functions. Descriptions of the special editing functions follow the table.

Table 4.1 Special Editing Commands

Command	Code	Function
Copy One character	<C1> F 5	Copy one character from the template to the new line
Copy up to character	<CM> F 6	Copy all characters from the template to the new line up to the character specified
Copy Template	<CT> F 7	Copy all remaining characters in the template to the new line
Skip One character	<S1> F 8	Do not copy (skip over) a character in the template
Skip up to character	<SM> F 9	Do not copy (skip over) the characters in the template up to the character specified
Quit Input	<QI>	Void the current input; leave the template unchanged
Insert mode	<INS> F 2	Enter insert mode
Replace mode	<REP> F 3	Exit insert mode (toggle from insert); this is the default
New Template	<NT>	Make the new line the new template

ESC [> 3 l

KEY

<Cl>

FUNCTION

Copy one character from the template to the input buffer.

COMMENTS

Pressing the <Cl> key copies one character from the template to the input buffer. When the <Cl> key is pressed, one character is inserted in the buffer and insert mode is automatically turned off if it was on. Use the <Cl> key to advance the cursor one column across the line.

EXAMPLE

Assume the screen shows:

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

At the beginning of the intraline edit, the cursor is positioned at the beginning of the line (indicated by the underline). Pressing the <Cl> key copies the first character (T) to the second of the two lines displayed:

```
     1:*This is a sample file
<Cl> 1:*T_
```

Each time the <Cl> key is pressed, one more character appears:

```
<Cl> 1:*Th_
<Cl> 1:*Thi_
<Cl> 1:*This_
```

KEY

<CM>

FUNCTION

Copy multiple characters up to a given character

COMMENTS

Pressing the <CM> key copies all characters up to a given character from the template to the input buffer. The given character is the next character typed and is not copied or shown on the screen. Pressing the <CM> key causes the cursor to move to the single character that is this command's only parameter. If the template does not contain the specified character, nothing is copied. Pressing <CM> also automatically turns off insert mode if it is on.

EXAMPLE

Assume the screen shows:

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

At the beginning of the intraline edit, the cursor is positioned at the beginning of the line (indicated by the underline). Pressing the <CM> key copies all characters up to the character pressed immediately after the <CM> key.

```
           1:*This is a sample file
     <CM>p 1:*This is a sam_
```

KEY

<CT>

FUNCTION

Copy template to input buffer

COMMENTS

Pressing <CT> copies all remaining characters from the template to the input buffer. Regardless of the cursor position at the time the <CT> key is pressed, the rest of the line appears, and the cursor is positioned after the last character on the line.

EXAMPLE:

Assume the screen shows:

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

At the beginning of the intraline edit, the cursor is positioned at the beginning of the line (indicated by the underline). Pressing the <CT> key copies all characters from the template (shown in the upper line displayed) to the line with the cursor (the lower line displayed):

```
          1:*This is a sample file
     <CT> 1:*This is a sample file._
```

Also, insert mode is automatically turned off if it was on.

KEY

<S1>

FUNCTION

Skip over one character in the template

COMMENTS

Pressing the <S1> key skips over one character in the template. Each time you press the <S1> key, one character is deleted (not copied) from the template. The action of the <S1> key is similar to the <C1> key, except that <S1> skips a character in the template rather than copies it to the input buffer.

EXAMPLE:

Assume the screen shows:

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

At the beginning of the intraline edit, the cursor is positioned at the beginning of the line (indicated by the underline). Pressing the <S1> key skips over the first character ("T").

```
     1:*This is a sample file
<S1> 1:*_
```

The cursor position does not move, only the template is affected. To see how much of the line has been (skipped over), press the <CT> key, which moves the cursor beyond the last character of the line.

```
     1:*This is a sample file.
<S1> 1:*_
<CT> 1:*his is a sample file._
```

KEY

<SM>

FUNCTION

Skip multiple characters in the template

COMMENTS

Pressing the <SM> key skips over all characters up to a given character in the template. The given character is the next character typed, and is not copied and not shown on the screen. If the template does not contain the specified character, nothing is skipped over. The action of the <SM> key is similar to the <CM> key, except that <SM> skips over characters in the template rather than copies them to the input buffer.

EXAMPLE

Assume the screen shows:

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

At the beginning of the intraline edit, the cursor is positioned at the beginning of the line (indicated by the underline). Pressing the <SM> key skips over (deletes) all the characters in the template up to the character pressed after the <SM> key:

```
           1:*This is a sample file
     <SM>p 1:*_
```

The cursor position does not move. To see how much of the line has been skipped over, press the <CT> key to copy the template. This moves the cursor beyond the last character of the line:

```
           1:*This is a sample file:
     <SM>p 1:*_
     <CT>  1:*ple file._
```

KEY

<QI>

FUNCTION

Quit input and flush the input buffer.

COMMENTS

Pressing the <QI> key flushes the input buffer, but it leaves the template unchanged. <QI> also prints a back slash (\), carriage return, and line feed, and turns insert mode off if it was on. The cursor is positioned at the beginning of the line. Pressing the <CT> key copies the template to the input buffer just as the line was before <QI> was pressed.

EXAMPLE

Assume the screen shows:

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

At the beginning of the intraline edit, the cursor is positioned at the beginning of the line (indicated by the underline). Assume you want to replace the line by typing:

```
                        1:*This is a sample file.
          Sample File 1:*Sample File_
```

Now, to reedit the line, press <QI>:

```
     1:*This is a sample file.
<QI> 1:*Sample File\
     1:_
```

<RETURN> can now be pressed to keep the original line or to perform any other intraline editing functions. If <CT> is pressed, the original template is copied to the input buffer:

```
<CT> 1: This is a sample file._
```

KEY

<INS>

FUNCTION

Enter insert mode

COMMENTS

Pressing the <INS> key causes entry into insert mode. The current position in the template is not changed. The cursor does move as each character is inserted. However, when you have finished inserting characters, the cursor is positioned at the same character as it was before the insertion began. Thus, characters are inserted <u>before</u> the character the cursor points to.

EXAMPLE

Assume the screen shows:

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

At the beginning of the intraline edit, the cursor is positioned at the beginning of the line (indicated by the underline). Assume you press the <CM> and "p" keys:

```
           1:*This is a sample file
     <CM>p 1:*This is a sam_
```

Now press the INS key and insert the three characters "s", "o", and "n".

```
              1:*This is a sample file.
     <CM>p    1:*This is a sam_
     <INS>son 1:*This is a samson_
```

If you now press the <CT> key, the rest of the template is copied to the line:

```
          1:*This is a samson_
     <CT> 1:*This is a samsonple file._
```

If you were to press the <RETURN> key, instead, the remainder of the template would be truncated, and the input buffer ended at the end of the insert:

```
     <INS>son<RETURN> 1:*This is a samson
```

KEY

<REP>

FUNCTION

Enter Replace mode.

COMMENTS

Pressing the <REP> key causes exit from insert mode and entry into replace mode. All characters entered overstrike and replace characters in the template. (Replace mode is the default.) When you start to edit a line, this mode is in effect. Each character typed replaces a character in the template. If the <RETURN> key is pressed, the remainder of the template is truncated.

EXAMPLE

Assume the screen shows:

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

At the beginning of the intraline edit, the cursor is positioned at the beginning of the line (indicated by the underline). Assume you then press <CM>p, <INS>son, <REP>ite, and then <CT>:

```
               l:*This is a sample file.
     <CM>p     l:*This is a sam_
     <INS>son  l:*This is a samson_
     <REP>ite  l:*This is a samsonite_
     <CT>      l:*This is a samsonite file._
```

If you type in characters that extend beyond the length of the template, the remaining characters in the template are automatically appended when you type <CT>.

KEY

<NT>

FUNCTION

Create new template

COMMENTS

Pressing the <NT> Key copies the current contents of the input buffer to the template. The contents of the old template are then destroyed. Pressing <NT> outputs an at sign character (@), a carriage return, and a line feed. The input buffer is also emptied and insert mode is turned off.

NOTE

<NT> performs the same functions as the <QI> key, except that the template is changed and an at sign character (@) is printed instead of a backslash (-).

EXAMPLE

Assume the screen shows:

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

At the beginning of the intraline edit, the cursor is positioned at the beginning of the line (indicated by the underline). Assume that you enter <CM>p, <INS>son, <REP>ite, and then <CT>:

```
            1:*This is a sample file.
<CM>p       1:*This is a sam_
<INS>son    1:*This is a samson_
<REP>ite    1:*This is a samsonite_
<CT>        1:*This is a samsonite file._
```

At this point, assume that you want this line as the new template, so you press the <NT> key:

```
<NT> 1:*This is a samsonite file.@

        _
```

Additional editing can now be done using the above new template.

4.3 INTERLINE COMMANDS

Interline commands perform editing functions on whole lines at a time. The interline commands are summarized in the following list and are described in detail with examples following the description of command parameters.

Table 4.1 Interline Commands

Command	Purpose
<line>	Edit Line
A	Append Lines
D	Delete Lines
E	End Editing
I	Insert Text
L	List Text
Q	Quit Editing
R	Replace Text
S	Search Text
W	Write Lines

4.3.1 Parameters

Each interline command accepts some optional parameters. The following list of parameters indicates their form. The effect of a parameter depends on the command it is used with.

PARAMETER DEFINITION

<line> <line> indicates a line number to be entered by the user. Line numbers must be separated from other line numbers, other parameters, and the command. Use a comma or space to separate.

<line> may be specified one of three ways:

Number any integer less than 65534. If a number larger than the largest existing line number is specified, then <line> indicates the line after the last line number.

Period (.) If a period is specified for <line>, then <line> indicates the current line number. The current line is the last line edited, and not necessarily the last line displayed. The current line is marked on your screen by an asterisk (*) between the line number and the first character.

Pound (#) The pound sign indicates the line after the last line number. Specifying # for <line> has the same effect as specifying a number larger than the last line number.

<RETURN> A carriage return entered without any of the <line> specifiers listed above directs EDLIN to use a default value appropriate to the command.

? The question mark parameter directs EDLIN to ask the user if the correct string has been found. The question mark is used only with the Replace and Search commands. Before continuing, EDLIN waits for either a "Y" or <RETURN> for a yes response, or for any other key for a no response.

<string> <string> represents text to be found, to be replaced, or to replace other text. The <string> parameter is used only with the Search and Replace commands. Each <string> must be terminated by a <CONTROL-Z> or a <RETURN> (see the Replace command for details). No spaces should be left between strings or between a string and its command letter, unless you want spaces as part of a string.

NAME

Edit

SYNTAX

[<line>]

FUNCTION

Edit line

COMMENTS

When a line number is entered, EDLIN displays the line number and text, then, on the line below, reprints the line number. The line is then ready for editing. You may use any of the available intraline commands to edit the line. The existing text of the line serves as the template until the <RETURN> key is pressed.

If no line number is entered (that is, only the <RETURN> key is pressed), the line after the current line, marked with an asterisk (*), is edited. If no changes of the current line are needed and the cursor position is at the beginning or end of the line, press the <RETURN> key to accept the line as is.

WARNING

If the <RETURN> key is pressed while the cursor is in the middle of the line, the remainder of the line is truncated.

EXAMPLE

Assume the following file exists and is ready to edit:

```
1: This is a sample file.
2: used to demonstrate
3: the editing of line
4:* four.
```

To edit line 4, enter:

```
4
```

The contents of the line are displayed along with a cursor below the line:

```
4:* four.
4:*_
```

Now type:

```
<INS>number  4: number_
<CT><RETURN> 4: number four.
             5:*_
```

NAME

Append

SYNTAX

```
[<n>]A
```

FUNCTION

Append lines from input file to editing buffer

COMMENTS

Use this command for extremely large files that will not fit into memory all at one time. By writing out part of the editing buffer to the output file with the Write command, room is made for lines to be appended with the Append command. If A is typed without a parameter, lines are appended to the part of the file currently in memory until available memory is 3/4 full or until there are no more lines to append.

Use the W command to write out lines to the output file. If the parameter <n> is given, then <n> lines are appended to that part of the file that currently is in memory. If <n> is not given, then as much of the input file as possible is read into the editing buffer until the editing buffer is three quarters full.

NAME

Delete

SYNTAX

```
[<line>][,<line>] D
```

FUNCTION

Delete the specified lines and all lines in between

COMMENTS

If the first <line> is omitted, the first <line> defaults to the current line (the line with the asterisk next to the line number). If the second <line> is omitted, then just the first <line> is deleted. When lines have been deleted, the line immediately after the deleted section becomes the current line and has the same line number as the first <line> had before the deletion occurred.

EXAMPLE

Assume the following file exists and is ready to edit:

```
 1: This is a sample file.
 2: Use: to demonstrate dynamic line numbers
 3: See what happens when you
 4: Delete and Insert
  .
  .
  .
25: (The D and I commands)
26: (Use <CONTROL-C> to exit insert mode)
27:*Line numbers
```

To delete multiple lines, enter <line>,<line> D:

```
5,24 D
```

The result is:

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3: See what happens when you
4: Delete and Insert
5*(The D and I commands)
6: (Use <CONTROL-C> to exit insert mode)
7: Line numbers
```

To delete a single line, enter:

```
6 D
```

The result is:

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3: See what happens when you
4: Delete and Insert
5: (The D and I commands)
6:*Line numbers
```

Next, delete a range of lines from the following file:

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3:*See what happens when you
4: Delete and Insert
5: (The D and I commands)
6: (Use <CONTROL-C> to exit insert mode)
7: Line numbers
```

To delete beginning with the current line, enter:

```
    ,6 D
```

The result is:

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3:*Line numbers
```

NAME

End

SYNTAX

E

FUNCTION

End the editing session

COMMENTS

Save the edited file on disk, rename the original input file "filename.BAK", and then exit EDLIN to the MS-DOS command level. If the file was created during the editing session, no .BAK file is created.

The E command takes no parameters. Therefore, you cannot tell EDLIN on which drive to save the file. The drive must be selected when the editing session is invoked. If the drive is not designated when EDLIN was invoked, the file is saved on the disk in the default drive. (It will still be possible to COPY the file to a different drive. However, this is done automatically if the drive is designated during invocation.)

You must be sure that the disk contains enough free space for the entire file to be written. If the disk does not contain enough free space, the write is aborted and the edited file lost, although part of the file may be written out.

EXAMPLE

The only possible command is:

```
E<RETURN>
```

After execution of the E command, control is returned to COMMAND.COM and the MS-DOS prompt is displayed.

NAME

Insert

SYNTAX

[<line>] I

FUNCTION

Insert line(s) of text immediately before the specified <line>.

COMMENTS

If you are creating a new file, the I command must be given before text can be inserted. In this case, the insert begins with line number 1.

EDLIN remains in insert mode until a <CONTROL-Z> or a <CONTROL-C> is entered. Successive line numbers appear automatically each time <RETURN> is pressed. When the insert is finished and insert mode has been exited, the <line>, which now immediately follows the inserted lines, becomes the current line. All line numbers following the inserted section are incremented by the number of lines inserted.

If <line> is not specified, the default is the current line number (the lines are inserted immediately before the current line). If <line> is an integer larger than the last line number, or if # is specified as <line>, the inserted lines are appended to the end of the file. In this case, the last line inserted becomes the current line. (This is the same as when the file is being created.)

EXAMPLE

Assume the following file exists and is ready to edit:

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3: See what happens when you
4: Delete and Insert
5: (The D and I commands)
6: (Use <CONTROL-C> to exit insert mode)
7:*Line numbers
```

To insert text before a specific line (not the current line), enter <line> I:

```
4 I
```

The result is:

```
4:_
```

Now, enter the new text for lines 4 and 5:

```
4: fool around with
5: those very useful commands that
```

Then to end the insertion, type:

```
6: <CONTROL-Z>
```

Now type L to list the file; the result is:

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3: See what happens when you
4: fool around with
5: those very useful commands that
6:*Delete and Insert
7: (The D and I commands)
8: (Use <CONTROL-C> to exit insert mode)
9: Line numbers
```

To insert lines immediately before the current line, enter:

```
I
```

The result is:

```
6: _
```

Now, insert the following text terminated with a <CONTROL-Z>:

```
6: perform the two major editing functions,
```

Now to List the file and see the result, type:

```
L
```

The result is:

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3: See what happens when you
4: fool around with
5: those very useful commands that
6: perform the two major editing functions,
7:*Delete and Insert
8: (The D and I commands)
```

```
 9: (Use <CONTROL-C> to exit insert mode)
10: Line numbers
```

To append new lines to the end of the file, enter:

```
11 I
```

This produces the following:

```
11: _
```

Now, enter the following new lines:

```
11: The insert command can place new lines
12: anywhere in the file; there's no space problems.
13: because the line numbers are dynamic;
14: They'll slide all the way to 65533.
```

End insertion by typing <CONTROL-C>. The new lines will appear at the end of all previous lines in the file. Now enter the list command:

```
L
```

The result is:

```
 1: This is a sample file.
 2: Use: to demonstrate dynamic line numbers
 3: See what happens when you
 4: fool around with
 5: those very useful commands that
 6: perform the two major editing functions,
 7: Delete and Insert
 8: (The D and I commands)
 9: (Use <CONTROL-C> to exit insert mode)
10: Line numbers
11: The insert command can place new lines
12: anywhere in the file; there's no space problems.
13: because the line numbers are dynamic;
14: They'll slide all the way to 65533.
```

NAME

List

SYNTAX

```
[<line>][,<line>] L
```

FUNCTION

List the specified range of lines, including the two lines specified.

COMMENTS

If the first <line> is omitted, the first <line> defaults to the current line. If the second <line> is omitted, 23 lines are listed; the eleven lines before <line>, the <line>, and the eleven lines after <line>. The current line remains unchanged. If the current line is one of the lines listed, it contains an asterisk between the line number and the first character.

EXAMPLE

Assume the following file exists and is ready to edit:

```
 1: This is a sample file.
 2: Use: to demonstrate dynamic line numbers
 3: See what happens when you
 4: Delete and Insert
 5: (The D and I commands)
  .
  .
  .
15:*The current line contains an asterisk.
  .
  .
  .
26: (Use <CONTROL-C> to exit insert mode)
27: Line numbers
```

To list a range of lines without reference to the current line, enter <line>,<line> L:

```
2,5 L
```

The result is:

```
2: Use: to demonstrate dynamic line numbers
3: See what happens when you
4: Delete and Insert
5: (The D and I commands)
```

To list a range of lines beginning with the current line, enter ,<line> L:

```
,26 L
```

The result is:

```
15:*The current line contains an asterisk.
   .
   .
   .
26: (Use <CONTROL-C> to exit insert mode)
```

To list a range of 23 lines around a specified line, enter <line>, L:

```
13, L
```

The result is:

```
13: The specified line is listed first in the range.
14: The current line remains unchanged by the L command.
15:*The current line contains an asterisk.
   .
   .
   .
35: <CONTROL-C> exits interline insert command mode.
```

To list a range of 23 lines centered around the current line, enter only L:

```
L
```

The result is:

```
 2: Use: to demonstrate dynamic line numbers
 3: See what happens when you
 4: Delete and Insert
 5: (The D and I commands)
  .
  .
  .
13: The current line is listed in the middle of the range.
14: The current line remains unchanged by the L command.
15:*The current line contains an asterisk.
  .
  .
  .
24: <CONTROL-C> exits interline insert command mode.
```

NAME

Quit

SYNTAX

```
Q
```

FUNCTION

Quit the editing session, do not save any editing changes, and exit to the MS-DOS operating system.

COMMENTS

No .BAK file is created. The Q command takes no parameters. It is simply a fast means of exiting an editing session. As soon as the Q command is given, EDLIN displays the message:

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

Press "Y" to quit the editing session;press "N" (or any other key except <CONTROL-C>) if you decide to continue the editing session.

EXAMPLE

Assume the following file exists and is ready to edit:

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3: Compare the before and after
4: See what happens when you
5: Delete and Insert
6: Line numbers
```

Now, to delete line 3, enter:

```
3 D
```

To list the file, enter "L":

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3: See what happens when you
4: Delete and Insert
5: Line numbers
```

Now, to keep the changes and to quit the editing session, enter:

```
Q
```

The result is:

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

Enter "Y" to exit to the operating system command level:

```
Abort edit (Y/N)?Y
A:_
```

NAME

Replace

SYNTAX

```
[<line>][,<line>] [?] R<string1><CONTROL-Z><string2>
```

FUNCTION

Replace all occurrences of <string1> in the specified range with <string2>.

COMMENTS

As each occurrence of <string1> is found, it is replaced by <string2>. Each line in which a replacement occurs is displayed. If a line contains two or more replacemments of <string1> with <string2>, then the line is displayed once for each occurrence. When all occurrences of <string1> in the specified range are replaced by <string2>, the R command terminates and the asterisk prompt reappears.

If a second string is to be given as a replacement, then <string1> must be terminated with a <CONTROL-Z>. If the <string2> is to be omitted, then <string1> may be terminated with either a combination <CONTROL-Z><RETURN>, or simply a <RETURN>. <String2> must also be terminated with a <CONTROL-Z><RETURN> combination or with a simple <RETURN>. If <string1> is omitted, then the replacement is terminated immediately. If <string2> is omitted, then <string1> is deleted from all lines in the range. If the first <line> is omitted in the range argument (as in ,<line>) then the first <line> defaults to the line after the current line. If the second <line> is omitted (as in <line> or <line>,) the second <line> defaults to #. Therefore, this is the same as <line>,#. Remember that # indicates the line after the last line of the file.

If the question mark (?) parameter is given, the Replace command stops at each line with a string that matches <string1>, displays the line with <string2> in place, and then displays the prompt "O.K.?". If the user presses "Y" or the <RETURN> key, then <string2> replaces <string1>, and the next occurrence of <string1> is found. Again, the "O.K.?" prompt is displayed. This process continues until the end of the range or until the end of the file. After the last occurrence of <string1> is found, EDLIN returns the asterisk prompt.

If you press any key besides "Y" or <RETURN> after the "O.K.?" prompt, the <stringl> is left as it was in the line, and the Replace goes to the next occurrence of <stringl>. If <stringl> occurs more than once in a line, each occurrence of <stringl> is replaced individually, and the "O.K.?" prompt is displayed after each replacement. In this way, only the desired <stringl> is replaced, and you prevent replacement of embedded strings.

EXAMPLE

Assume the following file exists and is ready for editing:

```
 1: This is a sample file.
 2: Use: to demonstrate dynamic line numbers
 3: See what happens when you
 4: fool around with
 5: those very useful commands that
 6: perform the two major editing functions,
 7: Delete and Insert
 8: (The D and I commands)
 9: (Use <CONTROL-C> to exit insert mode)
10: Line numbers
11: The insert command can place new lines
12: anywhere in the file; there's no space problems.
13: because the line numbers are dynamic;
14: They'll slide all the way to 65533.
```

To replace all occurrences of <stringl> with <string2> in a specified range, enter:

```
2,12 Rand<CONTROL-Z>or<RETURN>
```

The result is:

```
 5: those very useful commors that
 7: Delete or Insert
 8: (The D or I commands)
 8: (The D or I commors)
11: The insert commor can place new lines
```

Note that in the above replacement, some unwanted substitutions have occurred. To avoid these, and confirm each replacement, the same original file can be used:

```
         .
         .
      5: those very useful commands that
         .
      7: Delete and Insert
      8: (The D and I commands)
         .
     11: The insert command can place new lines
         .
         .
```

only with a slightly different command. This time, to replace only certain occurrences of the first <string> with the second <string>, enter:

```
      2? Rand<CONTROL-Z>or<RETURN>
```

The result is:

```
      5: those very useful commors that
      O.K.? N
      7: Delete or Insert
      O.K.? Y
      8: (The D or I commands)
      O.K.? Y
      8: (The D or I commors)
      O.K.? N
     11: The insert commor can place new lines
      O.K.? N
      *_
```

Now, enter the List command (L) to see the result of all these changes:

```
         .
         .
      5: those very useful commands that
         .
      7: Delete or Insert
      8: (The D or I commands)
         .
     11: The insert command can place new lines
         .
         .
```

NAME

Search

SYNTAX

```
[<line>][,<line>] [?] S<string>
```

FUNCTION

Search the specified range of lines for the specified string.

COMMENTS

The <string> must be terminated with a <RETURN>. The first line that matches <string> is displayed and becomes the current line. The Search command terminates when a match is found. If no line contains a match for <string>, the message "Not found" is displayed.

If the optional parameter, question mark (?), is included in the command, EDLIN displays the first line with a matching string; it then prompts the user with the message "O.K.?". If the user presses either the "Y" or <RETURN> key, the line becomes the current line and the search terminates. If the user presses any other key, the search continues until another match is found, or until all lines have been searched (then the "Not found" message is displayed).

If the first <line> is omitted (as in ,<line> S<string>), the first <line> defaults to the line after the current line. If the second <line> is omitted (as in <line> S<string> or <line>, S<string>), the second <line> defaults to #, which is the same as <line>,# S<string>. If <string> is omitted, no search is made and the command terminates immediately.

EXAMPLE

Assume the following file exists and is ready for editing:

```
1: This is a sample file.
2: Use: to demonstrate dynamic line numbers
3: See what happens when you
4: fool around with
5: those very useful commands that
6: perform the two major editing functions,
7: Delete and Insert
8: (The D and I commands)
```

```
 9: (Use <CONTROL-C> to exit insert mode)
10: Line numbers
11: The insert command can place new lines
12: anywhere in the file; there's no space problems.
13: because the line numbers are dynamic;
14:*They'll slide all the way to 65533.
```

To search for the first occurrence of a string, enter:

```
2,12 Sand<RETURN>
```

The result is:

```
 5: those very useful commands that
```

To get the "and" in line 7, modify the search command by entering:

```
<S1><CT>,12 Sand<RETURN>
```

The search then continues from the line after the current line (line 5), since no first line is given. The result is:

```
 7: Delete and Insert
```

To Search through several occurrences of a string until the correct string is found, enter:

```
1, ? Sand
```

The result is:

```
5: those very useful commands that
O.K.?_
```

If you press any key except "Y" or <RETURN>, the search continues, so type "N" here:

```
O.K.? N
```

Continue:

```
7 Delete and Insert
O.K.?_
```

Now press press "Y" to terminate the search:

```
O.K.? Y
*_
```

NAME

Write

SYNTAX

```
[<n>]W
```

FUNCTION

Write lines from the editing buffer to the output file

COMMENTS

The Write command is used when editing files that are larger than available memory. By executing the Write, lines are written out to the output file and room is made in the input buffer for more lines to be appended from the input file. If W is typed with no <n> parameter, then lines are written until memory is 1/4 full.

If the <n> parameter is given, then <n> lines are written out. Note that lines are written out beginning with the start of the file; subsequent lines in the editing buffer are renumbered beginning with one. A later Append command will append lines to any remaining lines in the editing buffer.

4.4 ERROR MESSAGES

EDLIN error messages occur either when you try to invoke EDLIN or during the actual editing session.

4.4.1 Errors When Invoking EDLIN

Cannot edit .BAK file--rename file

Cause: The user attempted to edit a file with the filename extension .BAK. .BAK files cannot be edited because the extension is reserved for backup copies.

Cure: If the user needs the .BAK file for editing purposes, the user must either RENAME the file with a different extension or COPY the .BAK file but with a different filename extension.

No room in directory for file

Cause: When the user attempted to create a new file, either the file directory was full or the user specified an illegal disk drive or an illegal filename.

Cure: Check the EDLIN invocation command line for illegal filename and illegal disk drive entries. If the command is no longer on the screen and if the user has not yet entered a new command, the EDLIN invocation command can be recovered by pressing the <CT> key.

If the invocation command line contains no illegal entries, run the CHKDSK program for the specified disk drive. If the status report shows the disk directory full, remove the disk and insert and format a new disk. If the CHKDSK status report shows the disk directory is not full, check the EDLIN invocation command for an illegal filename or illegal disk drive designation.

4.5 ERRORS WHILE EDITING

Entry Error

Cause: The last command entered contained a syntax error.

Cure: Reenter the command with the correct syntax.

Line too long

Cause: During Replace command mode, the string given as the replacement causes the line to expand beyond the limit of 254 characters. EDLIN aborts the Replace command.

Cure: Divide the long line into two lines, then retry the Replace command.

Disk Full--file write not completed

Cause: The user gave the End command, but the disk did not contain enough free space for the whole file. EDLIN aborts the E command and returns the user to the operating system. Some of the file may have been written to the disk.

Cure: Only a portion (at most) of the file will have been saved. The user should probably delete whatever file was saved and restart the editing session. None of the file not written out will be available after this error. Always be sure that the disk has sufficient free space for the file to be written, before you begin your editing session.

CHAPTER 5

DEBUG

DEBUG is a debugging program used to provide a controlled testing environment for binary and executable object files. Note that EDLIN is used to alter source files; DEBUG is EDLIN's counterpart for binary files. DEBUG eliminates the need to reassemble a program to see if a problem has been fixed by a minor change. It allows you to alter the contents of a file or the contents of a CPU register, and then to immediately reexecute a program to check of the validity of the changes.

All DEBUG commands may be aborted at any time by pressing <CONTROL-C>. <CONTROL-S> suspends the display, so that the user can read it before the output scrolls away. Entering any key other than <CONTROL-C> or <CONTROL-S> restarts the display. All of these commands are consistent with the control character functions available at the MS-DOS command level.

5.1 INVOCATION

To invoke DEBUG, enter:

```
DEBUG [<filespec> [<arglist>] ]
```

For example, if a <filespec> is specified, then the following is a typical invocation:

```
DEBUG LINK.EXE
```

DEBUG then loads FILE.EXE into memory starting at 100 hexadecimal in the lowest available segment. The BX:CX registers are loaded with the number of bytes placed into memory. The DEBUG prompt is a right angle bracket (>).

An <arglist> may be specified if <filespec> is present. These are filename parameters and switches that are to be passed to the program <filespec>. Thus, when <filespec> is loaded into memory, it is loaded as if it had been invoked with the command:

```
<filespec> <arglist>
```

Here, <filespec> is the file to be debugged, and the <arglist> is the rest of the command line that is used when <filespec> is invoked and loaded into memory.

If no <filespec> is specified, then DEBUG is invoked as follows:

```
DEBUG
```

DEBUG the returns with the prompt, signaling that it is ready to accept user commands. Since no filename has been specified, current memory, disk sectors, or disk files can be worked on by invoking later commands.

5.2 COMMANDS

Each DEBUG command consists of a single letter followed by one or more parameters. Additionally, the control character and the special editing functions described in Chapter 2, all apply inside DEBUG.

If a syntax error occurs in a DEBUG command, DEBUG reprints the command line and indicates the error with an up-arrow (^) and the word error.

For example:

```
dcs:100 cs:110
    ^ error
```

All commands and parameters may be entered in either upper or lower case. Any combination of upper and lower case may be used in commands.

The DEBUG commands are summarized in Table 5.1 and are described in detail with examples following the description of command parameters.

Table 5.1 DEBUG Commands

DEBUG Command	Function
C<range> <address>	Compare
D[<address> [L<value>]] D[<range>]	Dump
E<address> [<list>]	Enter
F<range> <list>	Fill
G[=<address> [<address>...]]	Go
H<address> <address>	Hex
I<value>	Input
L[<address> [<drive><record><record>]]	Load
M<range> <address>	Move
N<filespec>	Name
O<value> <byte>	Output
Q	Quit
R[<register-name>]	Register
S<range> <list>	Search
T[=<address>][<value>]	Trace
U[<address> [L<value>]] U[<range>]	Unassemble
W[<address> [<drive><record><record>]]	Write

5.3 PARAMETERS

As the summary above shows, all DEBUG commands accept parameters, except the Quit command. Parameters may be separated by delimiters (spaces or commas), but a delimiter is required only between two consecutive hexadecimal values. Thus, the following commands are equivalent:

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

Also, entries may be made in any combination upper or lower case.

PARAMETER DEFINITION

<drive> A one digit hexadecimal value to indicate which drive a file will be loaded from or written to. The valid values are 0-3. These values designate the drives as follows: 0=A:, 1=B:, 2=C:, 3=D:.

<byte> A two digit hexadecimal value to be placed in or read from an address or register.

<record> A 1 to 3 digit hexadecimal value used to indicate the logical record number on the disk and the number of disk sectors to be written or loaded. Logical records correspond to sectors. However their numbering differs since they represent the entire disk space.

<value> A hexadecimal value up to four digits used to specify a port number or the number of times a command should repeat its functions.

<address> A two part designation consisting of either an alphabetic segment register designation or a four digit segment address plus an offset value. The segment designation or segment address may be omitted, in which case the default segment is used. DS is the default segment for all commands except G, L, T, U, and W, for which the default segment is CS. All numeric values are hexadecimal.

For example:

```
CS:0100
04BA:0100
```

The colon is required between a segment designation (whether numeric or alphabetic) and an offset.

<range> Either two <address>s: e.g., <address> <address>; or one <address>, an L, and a <value>: e.g., <address> L <value> where <value> is the number of lines the command should operate on.

Examples:

```
CS:100 110
CS:100 L 10
```

The following is illegal:

```
CS:100 CS:110
       ^ error
```

The limit for <range> is 10000 hex. To specify a <value> of 10000 hex within four digits, enter 0000 (or 0).

<list> A series of <byte> values or of <string>s.line. <list> must be the last parameter on the command line.

Example:

```
fcs:100 42 45 52 54 41
```

<string> Any number of characters enclosed in quote marks. Quote marks may be either single (') or double("). Within <string>s, the opposite set of quote marks may be used freely as literals. If the delimiter quote marks must appear within a <string>, the quote marks must be doubled. For example, the following strings are legal:

```
'This is a "string" is okay.'
'This is a ''string'' is okay.'
```

However, this string is illegal:

```
'This is a 'string' is not.'
```

Similarly, these strings are legal:

```
"This is a 'string' is okay."
"This is a ""string"" is okay."
```

However, this string is illegal:

```
"This is a "string" is not."
```

Note that the double quotations are not necessary in the following strings:

```
"This is a ''string'' is not necessary."
'This is a ""string"" is not necessary.'
```

The ASCII values of the characters in the string are used as a <list> of byte values.

NAME

Compare

SYNTAX

```
C<range> <address>
```

FUNCTION

Compare the portion of memory specified by <range> to a portion of the same size beginning at <address>.

COMMENTS

If the two areas of memory are identical, there is no display and DEBUG returns with the MS-DOS prompt. If there are differences, they are displayed as:

```
<addressl> <bytel> <byte2> <address2>
```

EXAMPLE

The following commands have the same effect:

```
C100,200 300

    or

C100L100 300
```

Each command compares the block of memory from 100 to 200H with the block of memory from 300 to 400H.

NAME

Dump

SYNTAX

```
D[<address>[ L <value>]]
D[<range>]
```

FUNCTION

Display the memory contents of either a single address, a range of addresses, or the number of lines specified by <value> beginning at the <address> specified.

COMMENTS

If a single address only is specified, the contents of 128 bytes are displayed. If a range of addresses is specified, the contents of the range are displayed. If the D command is entered without parameters, the result is the same as if the user specified a single address, except that the display begins at the current location in the DS (data) segment.

The dump is displayed in two portions: a hexadecimal dump (each byte is shown in hexadecimal value) and an ASCII dump (the bytes are shown in ASCII characters). Nonprinting characters are denoted by a period (.) in the ASCII portion of the display. Each display line shows sixteen bytes with a hyphen between the eighth and ninth bytes. At times, displyas are split in this manual to fit them on the page.

Each displayed line begins on a 16-byte boundary and the first line contains fewer bytes than the rest of the displayed lines.

EXAMPLE

Assume the following command is entered:

```
dcs:100 110
```

DEBUG displays:

```
04BA:0100  42 45 52 54 41 20 54 00
                          BERTA T.
04BA:0108  20 42 4F 52 4C 41 4E 44
                           BORLAND
```

If you enter the command:

```
dcs:100 110
```

DEBUG displays:

```
04BA:0100 42 45 52 54 41 ... 4E 44 BERTA T. BORLAND
```

If the following command is entered:

```
D
```

the display is formatted as described above. Each line of the display begins with an address; incremented by 16 from the address on the previous line. Each subsequent D (entered without parameters) displays the bytes immediately following those last displayed.

If the user enters the command:

```
DCS:100 L 20
```

the display is formatted as described above, but all of the bytes for 20H lines are displayed.

If the user enters the command:

```
DCS:100 115
```

the display is formatted as described above, but all the bytes in the range of lines from 100H to 115H in the CS segment are displayed.

NAME

Enter

SYNTAX

E<address>[<list>]

FUNCTION

Enter the <address>, display its contents, and wait for the user's input.

COMMENTS

If the optional <list> of hexadecimal values is entered, the replacement of byte values occurs automatically. (If an error occurs, no byte values are changed.) If the <address> is entered without the optional <list>, DEBUG displays the address and its contents, then repeats the address on the next line and waits for the user's input. At this point, the Enter command waits you to perform one of the following actions:

1. Replace a byte value with a value the user types in. The user simply types the value after the current value. If the value typed in is not a legal hexadecimal value or if more than two digits are typed, a bell sounds and the illegal or extra character is not echoed.

2. Press the space bar to advance to the next byte. To change the value, simply enter the new value as described in (1.) above. If the user spaces beyond an eight-byte boundary, DEBUG starts a new display line with the address displayed at the beginning.

3. Type a hyphen (-) to return to the preceding byte. If the user decides to change a byte behind the current position, typing the hyphen returns the current position to the previous byte. When the hyphen is typed, a new line is started with the address and its byte value displayed.

4. Press the <RETURN> key to terminate the Enter command. The <RETURN> key may be pressed at any byte position.

EXAMPLE

Assume the following command is entered:

```
ECS:100
```

DEBUG displays:

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

To change this value to 41, enter "41" as shown:

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

To step through the subsequent bytes, press the space bar to see:

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

To change BC to 42:

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

Now, realizing that 10 should be 6F; enter the hyphen as many times as needed to return to byte 0101 (value 10), then replace 10 with 6F:

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

Pressing the <RETURN> key ends the Enter command and returns to the DEBUG command level.

NAME

Fill

SYNTAX

```
F<range> <list>
```

FUNCTION

Fill the addresses in the <range> with the values in the <list>.

COMME'TS

If the <range> contains more bytes than the number of values in the <list>, the <list> will be used repeatedly until all bytes in the <range> are filled. If the <list> contains more values than the number of bytes in the <range>, the extra values in the <list> will be ignored. If any of the memory in the <range> is not valid (bad or nonexistent), the error will be propagated into all succeeding locations. The F command does not abort as the E command does. The F command is a multiple version of the E command in that it allows the user to change more than one address at a time.

EXAMPLE

Assume the following command is entered:

```
F04BA:100 L 100 42 45 52 54 41
```

DEBUG fills memory locations 04BA:100 through 04BA:200 with the bytes specified. The five values are repeated until all 100H bytes are filled.

NAME

Go

SYNTAX

```
G[=<address>[ <address>...]]
```

FUNCTION

Execute the program currently in memory.

COMMENTS

If the Go command is entered alone, the program executes as if the program had run outside DEBUG.

If =<address> is set, execution begins at the address specified. If the segment designation is omitted from =<address>, only the instruction pointer is set. If the segment designation is included in =<address>, both the CS segment and the instruction pointer are set. The equal sign (=) is required, so that DEBUG can distinguish the start =<address> from the breakpoint <address>es.

With the other optional addresses set, execution stops at the first <address> encountered, regardless of that address' position in the list of addresses to halt execution, no matter which branch the program takes. When program execution reaches a breakpoint, the registers, flags, and decoded instruction are displayed for the last instruction executed. (The result is the same as if you had entered the Register command for the breakpoint address.)

Up to ten breakpoints may be set. Breakpoints may be set only at addresses containing the first byte of an 8086 opcode. If more than 10 breakpoints are set, DEBUG returns the BP Error message.

The user stack pointer must be valid and have six bytes available for this command. The G command uses an IRET instruction to cause a jump to the program under test. The user stack pointer is set, and the user Flags, Code Segment register, and Instruction Pointer are pushed on the the user stack. (Thus, if the user stack is not valid or is too small, the operating system may crash.) An interrupt code (0CCH) is placed at the specified breakpoint address(es). When an instruction with the breakpoint code is encountered, all breakpoint addresses are restored to their original instructions. If execution is not halted at one of the breakpoints, the interrupt codes are not replaced with the original instructions.

EXAMPLE

Assume the following command is entered:

```
GCS:7550
```

The program currently in memory executes up to the address 7550 in the CS segment. Then DEBUG displays registers and flags, after which the Go command is terminated.

After a breakpoint has been encounterd, if you enter the Go command again, then the program executes just as if the user had entered the filename at the MS-DOS command level. The only difference is that program execution begins at the instruction after the breakpoint rather than at the usual start address.

NAME

Hex

SYNTAX

```
H<address> <address>
```

FUNCTION

Perform hexadecimal arithmetic on the two parameters.

COMMENTS

First, DEBUG adds the two parameters, then subtracts the second parameter from the first. The results of the arithmetic is displayed on one line; first the sum, then the difference.

EXAMPLE

Assume the following command is entered:

```
H10A 19F
```

DEBUG performs the calculations and then returns the results:

```
02A9   0095
```

NAME

Input

SYNTAX

I<value>

FUNCTION

Input and display one byte from the port specified by <value>.

COMMENTS

A 16-bit port address is allowed.

EXAMPLE

Assume the following command is entered:

```
I2F8
```

Assume also that the byte at the port is 42H. DEBUG inputs the byte and displays the value:

```
42
```

NAME

Load

SYNTAX

```
L[<address> [<drive> <record> <record>]]
```

FUNCTION

Load a file into memory.

COMMENTS

Set BX:CX to the number of bytes read. The file must have been named either with the DEBUG invocation command or with the N command. Both the invocation and the N commands format a filename properly in the normal format of a file control block at CS:5C.

If the L command is given without any parameters, DEBUG loads the file into memory beginning at address CS:100 and sets BX:CX to the number of bytes loaded. If the L command is given with an address parameter, loading begins at the memory <address> specified. If L is entered with all parameters, absolute disk sectors are loaded, not a file. The <record>s are taken from the <drive> specified (the drive designation is numeric here--0=A:, 1=B:, 2=C:, 3=D:); DEBUG begins loading with the first <record> specified; and continues until the number of sectors specified in the second <record> have been loaded.

EXAMPLE

Assume the following commands are entered:

```
A:DEBUG
>NFILE.COM
```

Now, to load FILE.COM, enter:

```
L
```

DEBUG loads the file and returns the DEBUG prompt. Assume you want to load only portions of a file or certain records from a disk. To do this, enter:

```
L04ba:100 2 0F 6D
```

DEBUG then loads 109 (6D hex) records beginning with logical record number 15 into memory beginning at address 04BA:0100. When the records have been loaded, DEBUG simply returns the its prompt.

If the file has a .EXE extension, then it is relocated to the load address specified in the header of the .EXE file: the <address> parameter is always ignored for .EXE files. Note that the header itself is stripped off the .EXE file before it is loaded into memory. Thus the size of a .EXE file on disk will differ from its size in memory.

If the file named by the Name command or specified on invocation is a .HEX file, then entering the L command with no parameters causes loading of the file beginning at the address specified in the .HEX file. If the L command includes the option <address>, DEBUG adds the <address> specified in the L command to the address found in the .HEX file to determine the start address for loading the file.

NAME

Move

SYNTAX

```
M<range> <address>
```

FUNCTION

Move the block of memory specified by <range> to the location beginning at the <address> specified.

COMMENTS

Overlapping moves (moves where part of the block overlaps some of the current addresses) are always performed without loss of data. Addresses that could be overwritten are moved first. The sequence for moves from higher addresses to lower addresses is to move the data beginning at the block's lowest address and working towards the highest. The sequence for moves from lower addresses to higher addresses is to move the data beginning at the block's highest address and working towards the lowest.

Note that if the addresses in the block being moved will not have new data written to them, the data there before the move will remain; that is, the M command really copies the data from one area into another, in the sequence described, and writes over the new addresses. This is why the sequence of the move is important.

EXAMPLE

Assume you enter:

```
MCS:100 110 CS:500
```

DEBUG first moves address CS:110 to address CS:510, then CS:10F to CS:50F, and so on until CS:100 is moved to CS:500. You should enter the D command, using the <address> entered for the M command, to review the results of the move.

NAME

Name

SYNTAX

```
N<filename>[<filename>...]
```

FUNCTION

Set filenames

COMMENTS

The Name command performs two distinct functions, both having to do with filenames. First, Name is used to assign a filename for a later Load or Write command. Thus, if you invoke DEBUG without naming any file to be debugged, then the N<filename> command must be given before a file can be Loaded. Second, Name is used to assign filename parameters to the file being debugged. In this case, Name accepts a list of parameters that are used by the file being debugged.

These functions overlap. Consider the following set of DEBUG commands:

```
>NFILE1.EXE
>L
>G
```

Because of the two-pronged effect of the Name command, the following happens:

1. (N)ame assigns the filename FILE1.EXE to the filename to be used in any later Load or Write commands.

2. (N)ame also assigns the filename FILE.EXE to the first filename parameter to be used by any program that is later debugged.

3. (L)oad loads FILE.EXE into memory.

4. (G)o causes FILE.EXE to be executed with FILE.EXE as the single filename parameter (that is, FILE.EXE is executed as if FILE FILE.EXE had been typed at the command level.

A more useful chain of commands might go like this:

```
>NFILE1.EXE
>L
>NFILE2.DAT FILE3.DAT
>G
```

Here, Name sets FILE1.EXE as the filename for the subsequent Load command. The Load command loads FILE1.EXE into memory, and then the Name command is used again, this time to specify the parameters to be used by FILE1.EXE. Finally, when the Go command is executed, FILE1.EXE is executed as if FILE1 FILE2.DAT FILE3.DAT had been typed at the MS-DOS command level. Note that if a Write command were executed at this point, then FILE1.EXE--the file being debugged--would be saved with the name FILE2.DAT! To avoid such undesired results, you should always execute a Name command before either a Load or a Write.

There are four distinct regions of memory that can be affected by the Name command:

CS:5C	FCB for file 1
CS:6C	FCB for file 2
CS:80	Count of characters
CS:81	All characters entered

A File Control Block (FCB) for the first filename parameter given to the Name command is set-up at CS:5C. If a second filename parameter is given, then an FCB is setup for it beginning at CS:6C. The number of characters typed in the Name command (exclusive of the first character, "N") is given at location CS:80. The actual stream of characters given by the Name command (again, exclusive of the letter "N") begins at CS:81. Note that this stream of characters may contain switches and and delimiters that would be legal in any command typed at the MS-DOS command level.

EXAMPLE

A typical use of the Name command would be:

```
DEBUG PROG.COM
-NPARAM1 PARAM2/C
-G
-
```

In this case, the Go command executes the file in memory as if the following command line had been entered:

```
PROG PARAM1 PARAM2/C
```

Testing and debugging therefore reflect a normal runtime environment for PROG.COM.

NAME

Output

SYNTAX

```
O<value> <byte>
```

FUNCTION

Send the <byte> specified to the output port specified by <value>.

COMMENTS

A 16-bit port address is allowed.

EXAMPLE

Enter:

```
O2F8 4F
```

DEBUG outputs the byte value 4F to output port 2F8.

NAME

Quit

SYNTAX

Q

FUNCTION

Terminate the debugger.

COMMENTS

The Q command takes no parameters and exits DEBUG without saving the file currently being operated on. You are returned to the MS-DOS commands level.

EXAMPLE

To end the debugging session, enter:

```
    Q<RETURN>
```

DEBUG is terminated, and control returns to the MS-DOS command level.

NAME

Register

SYNTAX

R[<register-name>]

FUNCTION

Display the contents of one or more CPU registers.

COMMENTS

If no <register-name> is entered, the R command dumps the register save area and displays the contents of all registers and flags.

If a register name is entered, the 16-bit value of that register is displayed in hexadecimal, and then a colon appears as a prompt. The user then either enters a <value> to change the register, or simply presses the <RETURN> key if no change is wanted.

The only valid <register-name>s are:

```
AX    BP    SS
BX    SI    CS
CX    DI    IP    (IP and PC both refer
DX    DS    PC     to the instruction
SP    ES    F      pointer.)
```

Any other entry for <register-name> results in a BR Error message.

If F is entered as the <register-name>, DEBUG displays a two character alphabetic code. To alter the flag, enter the opposite two letter code. The flags are either set or clear.

The flags with their codes for set and clear are listed below:

FLAG NAME	SET	CLEAR
Overflow	OV	NV
Direction	DN Decrement	UP Increment
Interrupt	EI Enabled	DI Disabled
Sign	NG Negative	PL Plus
Zero	ZR	NZ
Auxiliary Carry	AC	NA
Parity	PE Even	PO Odd
Carry	CY	NC

Whenever the user enters the command RF, the flags are displayed in the order shown above in a row at the beginning of a line. At the end of the list of flags, DEBUG displays a static hyphen (-); then the DEBUG prompt (>). Your may enter new flag values as alphabetic pairs. The new flag values can be entered in any order. You are not required to leave spaces between the flag entries. To exit the R command, press the <RETURN> key. Flags for which new values were not entered remain unchanged.

If more than one value is entered for a flag, DEBUG returns a DF Error message. If the you enter a flag code other than those shown above, DEBUG returns a BF Error message. In both cases, the flags up to the error in the list are changed; flags at and after the error are not.

At start up, the segment registers are set to the bottom of free memory, the Instruction Pointer is set to 0100H, the Stack Pointer is set to 005AH, all flags are cleared, and the remaining registers are set to zero.

EXAMPLE

Enter:

```
     R
```

DEBUG displays all registers, flags, and the decoded instruction for the current location. If the location is CS:11A, then DEBUG might display:

```
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
```

If you enter:

```
     RF
```

DEBUG displays the flags:

```
     NV UP DI NG NZ AC PE NC - _
```

Now enter any valid flag designation, in any order, with or without spaces.

For example:

```
     NV UP DI NG NZ AC PE NC - PLEICY<RETURN>
```

DEBUG responds only with the DEBUG prompt. To see the changes, enter either the D, R, or RF command:

```
     RF
     NV UP EI PL NZ AC PE CY - _
```

Press <RETURN> to leave the flags this way, or to enter different flag values.

NAME

Search

SYNTAX

```
S<range> <list>
```

FUNCTION

Search the range specified for the <list> of bytes specified.

COMMENTS

The <list> may contain one or more bytes, each separated by a space or comma. If the <list> contains more than one byte, only the first address of the byte string is returned. If the <list> contains only one byte, all addresses of the byte in the <range> are displayed.

EXAMPLE

If you enter:

```
SCS:100 110 41
```

DEBUG might return the response:

```
04BA:0104
04BA:010D
>_
```

NAME

Trace

SYNTAX

```
T[=<address>][ <value>]
```

FUNCTION

Execute one instruction and display the contents of all registers, flags, and the decoded instruction.

COMMENTS

If the optional =<address> is entered, tracing occurs at the =<address> specified. If =<address> includes the segment designation, then both CS and the instruction pointer are specified. If =<address> omits the segment designation, only the instruction pointer is specified. The optional <value> causes DEBUG to execute and trace the number of steps specified by <value>.

The T command uses the hardware trace mode of the 8086 or 8088 microprocessor. Consequently, the user may also trace instructions stored in ROM.

EXAMPLE

Enter:

```
T
```

DEBUG returns a display of the registers, flags, and decoded instruction for that one instruction. Assume that the current position is 04BA:011A; then DEBUG might return the display:

```
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
```

Now enter:

```
T=011A 10
```

DEBUG executes ten instructions beginning at 011A in the current segment and then displays all registers and flags for each instruction as it is executed. The display scrolls away until the last instruction is executed. Then the display stops, and you can see the register and flag values for the last few instructions performed. Remember that <CONTROL-S> suspends the display at any point, so that you can study the registers and flags for any instruction.

NAME

Unassemble

SYNTAX

```
U[<address>[ L <value>]
U[<range>]
```

FUNCTION

Disassemble bytes and display the source statements that correspond to them, along with addresses and byte values.

COMMENTS

The display of disassembled code looks like a listing for an assembled file. If you enter the U command without parameters, 20 hexadecimal bytes are disassembled to show corresponding instructions. If you enter the U command with the <range> parameter, then DEBUG disassembles all bytes in the range up to 20 hexadecimal bytes. If there are fewer bytes in the <range> than the number displayed when U is entered without parameters, then only the bytes in the <range> are displayed.

If you enter the U command with the <address> parameter, then DEBUG disassembles the default number of bytes, beginning at the <address> specified. If you enter the U command with the <address> L <value> parameters, then DEBUG disassembles all bytes beginning at the address specified for the number of lines specified by <value>. Entering the U command with the <address> L <value> parameters overrides the default limit (20H bytes).

EXAMPLE

Enter:

```
U04BA:100
```

DEBUG disassembles 16 bytes beginning at address 04BA:0100:

```
    04BA:0100  206472     AND  [SI+72],AH
    04BA:0103  69         DB   69
    04BA:0104  7665       JBE  016B
    04BA:0106  207370     AND  [BP+DI+70],DH
    04BA:0109  65         DB   65
    04BA:010A  63         DB   63
    04BA:010B  69         DB   69
    04BA:010C  66         DB   66
    04BA:010D  69         DB   69
    04BA:010E  63         DB   63
    04BA:010F  61         DB   61
```

If you enter:

```
     u04ba:0100 0108
```

The display shows:

```
     04BA:0100  206472     AND  [SI+72],AH
     04BA:0103  69         DB   69
     04BA:0104  7665       JBE  016B
     04BA:0106  207370     AND  [BP+DI+70],DH
```

However, if you enter:

```
     u04ba:100 120
```

Then the display appears exactly the same as above with the UCS:100 command.

If, however, you enter the command:

```
     UCS:100 L 20
```

all of the bytes in the twenty lines of instructions beginning at address CS:100 through address CS:120 are disassembled and displayed. This applies to both display formats.

If the bytes in some addresses are altered, the disassembler alters the instruction statements. The U command can be entered for the changed locations, the new instructions viewed, and the disassembled code used to edit the source file.

NAME

Write

SYNTAX

```
W[<address>[ <drive> <record> <record>]]
```

FUNCTION

Write the file being debugged to a disk file.

COMMENTS

If only the W appears, BX:CX must already be set to the number of bytes to be written; the file is written beginning from CS:100. If the W command is given with just an address, then the file is written beginning at that address. If a G or T command was used, BX:CX must be reset before using the Write command without parameters. (Note that if a file is loaded and modified, the name, length, and starting address are all set correctly to save the modified file as long as the length has not changed.)

The file must have been named either with the DEBUG invocation command or with the N command (see Name above). Both the invocation and the N commands format a file name properly in the normal format of a file control block at CS:5C.

If the W command is given without parameters, BX:CX must be set to the number of bytes to be written. Then, DEBUG writes the BX:CX number of bytes to the disk file. The debugged file is written to the disk from which it was loaded. This means that the debugged file is written over the original file that was loaded into memory.

If the W command is given with parameters, the write begins from the memory address specified; the file is written to the <drive> specified (the drive designation is numeric here--0=A:, 1=B:, 2=C:, 3=D:); DEBUG writes the file beginning at the logical record number specified by the first <record>; and continues until the number of sectors specified in the second <record> have been written.

WARNING

Writing to absolute sectors is EXTREMELY dangerous because the process bypasses the file handler.

EXAMPLE

Enter:

```
W
```

DEBUG writes out the file to disk then displays the DEBUG prompt:

```
W
>_
```

Another example:

```
WCS:100 1 37 2B
```

DEBUG writes out the contents of memory, beginning with the address CS:100 to the disk in drive B:. The data written out starts in disk logical record number 37H and consists of 2BH records. When the write is complete, DEBUG displays the prompt:

```
WCS:100 1 37 2B
>_
```

5.4 ERROR MESSAGES

During the DEBUG session, you may receive any of the following error messages. Each error terminates the DEBUG command with which it is associated, but does not terminate DEBUG itself.

ERROR CODE DEFINITION

BF Bad Flag
The user attempted to alter a flag, but the characters entered were not one of the acceptable pairs of flag values. See the Register command for the list of acceptable flag entries.

BP Too many Breakpoints
The user specified more than ten breakpoints as parameters to the G command. Reenter the Go command with ten or fewer breakpoints.

BR Bad Register
The user entered the R command with an invalid register name. See the Register command for the list of valid register names.

DF Double Flag
The user entered a two values for one flag. The user may specify a flag value only once per RF command.

CHAPTER 6

FILCOM

The FILCOM File Comparison Utility compares the contents of two files. The differences between the two files are output to a third file. The files being compared may be either source files (files containing source statements of a programming language) or binary files (files output by the MACRO-86 assembler, the MS-LINK Linker Utility, or by one of the Microsoft high-level language compilers).

Limitations on Source Comparisons

FILCOM uses all available memory as buffer space to hold the source files. If the source files are larger than available memory, FILCOM comparisons what it is able to load into the buffer space. If no matches are found within the portions of the files in the buffer space, FILCOM outputs only the message:

FILES ARE DIFFERENT

For binary files larger than available memory, FILCOM compares both files completely, overlaying the portion in memory with the next portion from disk. All differences are output the same as for binary comparisons of files that fit completely in memory.

6.1 INVOCATION

FILCOM can be invoked in one of two ways.

Method 1:

Enter:

```
     FILCOM
```

FILCOM responds with the first prompt and then waits:

```
     Source 1 Filename [.ASM]:
```

Method 2:

Enter:

```
     FILCOM <sourcel>,[<source2>],[<list>][/<switch>...]
```

FILCOM and the filenames must be separated by commas. The slash mark is the only delimiter allowed between a filename and a switch letter. Switches may be placed after any of the entries in the invocation command line, but before the comma.

If you want to select the default for Source 2 but not for List, enter two consecutive commas between Source 1 and List entries.

For example:

```
     FILCOM ALPHA,,GAMMA
```

When method 2 is used, FILCOM responds with a banner but no prompts, and performs the comparison. Method 2 permits FILCOM commands to be used in a batch file under MS-DOS, as well as permitting you to enter all commands on one line at one time. When FILCOM is finished, the operating system prompt reappears.

6.2 COMMANDS

Commands to FILCOM consist of responses to three prompts for file specifications, plus optional switches. The file specifications may be entered one at a time as the prompts appear, or all at once as part of the FILCOM invocation command (method 2).

6.2.1 File Specifications

All file specifications take the form:

```
d:filename.ext
```

where: d: is the letter of a disk drive. If the drive designation is omitted, FILCOM defaults to the operating system's (current) default drive.

filename is a 1-8 character name of the file.

.ext is a 1-3 character extension to the filename. See Section 5.2.3, Defaults and Shortcuts, for a list of the default filename extensions used under FILCOM and their effects.

6.2.2 Prompts

If invocation method 2 is used, FILCOM displays no prompts; it simply performs the comparison and exits to the operating system.

If invocation method 1 is used (or else method 2 with an illegal filename or the name of a nonexistent file for the first source file), FILCOM displays the first prompt:

```
     Source 1 Filename [.ASM]:_
```

Enter the name of one of the files you want compared. If the filename extension for this file is .ASM, the extension may be omitted from the entry. Otherwise, the extension must be included.

When a legal response has been entered, FILCOM responds with the second prompt:

```
     Source 2 Filename [sourcel.BAK]:_
```

Enter the name of the file you want compared to Source 1. FILCOM defaults to the backup file for the file named for the first prompt. If the response to prompt 1 is TEST (meaning TEST.ASM), FILCOM will display as default for Source 2 the filename TEST.BAK. If you want to compare the Source 1 file with its backup file, simply press <RETURN>. Otherwise, enter a filename. Likewise, if the Source 2 file has a filename extension of .BAK, the extension may be omitted. Otherwise, the extension must be entered, too.

When a legal response to the second prompt has been entered, FILCOM responds with the third prompt:

```
     List Filename [sourcel.DIF]:_
```

Enter the name of the file to receive the list of differences. FILCOM defaults to the name given for Source 1 with a default filename extension of .DIF. Again, if the response to prompt 1 was TEST (meaning TEST.ASM), FILCOM displays TEST.DIF as the default List filename. If this default filename is acceptable to you for the List file, simply press the carriage return key. Otherwise, enter the filename. Likewise, if the filename extension .DIF is acceptable to you, the extension may be omitted, even if you do specify a filename. If .DIF is an unacceptable filename extension, enter an extension along with the filename.

When FILCOM is finished comparing the two source files and has output the differences to the List file, the operating system prompt reappears.

6.2.3 Defaults And Shortcuts

FILCOM recognizes the following default extensions:

Prompt	Extension	Effect
Source 1	.ASM	Default Source 1 filename extension. May be overridden.
	.OBJ .EXE .COM	Causes default to binary comparison
Source 2	.BAK	Default Source 2 Filename extension. May be overridden.
List	.DIF	Default List Filename extension. May be overridden.

Shortcuts

Two shortcuts for entering commands are supported. Both shortcuts use default responses for any prompts to which a response is not entered.

Carriage return key

The Source 1 Filename [.ASM] prompt requires at least a filename response.

The Source 2 Filename [sourcel.BAK] and List Filename [sourcel.DIF] prompts show a default entry; the filename entered for Source 1 and a default filename extension. To select the default entry, simply press the <RETURN> key.

Example:

```
Source 1 Filename [.ASM]: TEST
Source 2 Filename [TEST.BAK]: <RETURN>
List Filename [TEST.DIF]: <RETURN>
```

These responses cause FILCOM to compare TEST.ASM with TEST.BAK and to output any differences in the file TEST.DIF.

<RETURN> only may be entered for either of the prompts, regardless of what you plan to enter for the other. For example, the <RETURN> may be used to select the default for Source 2, yet allows a nondefault entry for List.

Example:

```
Source 1 Filename [.ASM]: TEST
Source 2 Filename [TEST.BAK]: <RETURN>
List Filename [TEST.DIF]: PAST.PRN
```

These responses cause FILCOM to compare TEST.ASM with TEST.BAK and to output any differences in the file PAST.PRN (default for Source 2 was selected, but not for List.)

Semicolon (;)

The semicolon character (;) also selects the default responses to the Source 2 and List prompts. If the semicolon is entered following the Source 2 Filename prompt, the List Filename prompt will not appear. That is, the semicolon selects the default response for all remaining prompts. Unlike the carriage return key, once you enter semicolon, comparing begins, and you have no chance to enter another response for that comparison. Indeed, you may think of the semicolon as a message to FILCOM that you want to use all default responses only.

This is especially useful when using invocation method 2:

Example:

```
FILCOM TEST;
```

This entry causes FILCOM to display its banner and then perform the desired comparison. FILCOM compares TEST.ASM with TEST.BAK and outputs the differences in the file TEST.DIF (the same as the example under "Carriage Return Key" above).

Now, consider this sample invocation:

```
FILCOM
Source 1 Filename [.ASM]: TEST;
```

This set of commands and responses produces the same result as the two previous examples. Note that you have no chance to enter alternatives to the defaults for Source 2 or List when you use the semicolon.

6.2.4 Switches

FILCOM supports five switches. Switches are single letters appended to the (method 2) invocation command line or to any of the prompt responses to control the file comparison. A switch must always be preceded by a slash.

FILCOM switches are one of two types: source comparison or binary comparison.

Source Comparison Switches

/A Force a source comparison of files with filename extensions .OBJ, .EXE, and .COM. FILCOM defaults to binary comparison on files with these filename extensions. Files with any other filename extensions default to source comparison. Therefore, /A is not required for files that do not have one of these three filename extensions.

/C Include comments in comparison. A comment starts with a semicolon (;), and ends with an end-of-line character. By default, comments are not included in comparisons. Thus, only functional changes in source files are detected by FILCOM. This means that comments in two files, even if different, are ignored when searching for consecutive lines that match (see the /<n> switch below for an explanation of "match").

/<n> <n> is a number from 1 through 9. Default is 3. <n> specifies how many consecutive lines in the two files must be the same before FILCOM considers that the two files match at that point. (Refer to examples 1 and 2 below for a demonstration of the effects of the /<n> switch).

When FILCOM finds <n> lines that match, it outputs the lines that are different since the last <n> lines that matched, plus the first line of the current <n> lines that match. The first match line should help locate where differences occurred.

/S Include spaces and tabs in comparisons. By default, spaces and tabs are <u>not</u> included in comparisons. Thus, only functional changes in source files are detected by FILCOM. This means that spaces and tabs in the two files, even if the same, are ignored and are not used to find matches.

Binary Comparison Switch

/B Force binary comparison of files that default to source comparison (files without the filename extensions .OBJ, .EXE, or .COM). Instead of a source comparison of lines, FILCOM compares the two files byte-by-byte. For differences, FILCOM outputs the offset location into the files and the differing bytes in hexadecimal. (Refer to example 3 for a demonstration.)

6.3 EXAMPLES

Example 1

Assume these two ASCII files are on disk:

```
         ALPHA.ASM        BETA.ASM

         FILE A           FILE B
         ________________________
         A                A
         B                B
         C                C
         D                G
         E                H
         F                I
         G                J
         H                1
         I                2
         M                P
         N                Q
         O                R
         P                S
         Q                T
         R                U
         S                V
         T                4
         U                5
         V                W
         W                X
         X                Y
         Y                Z
         Z
```

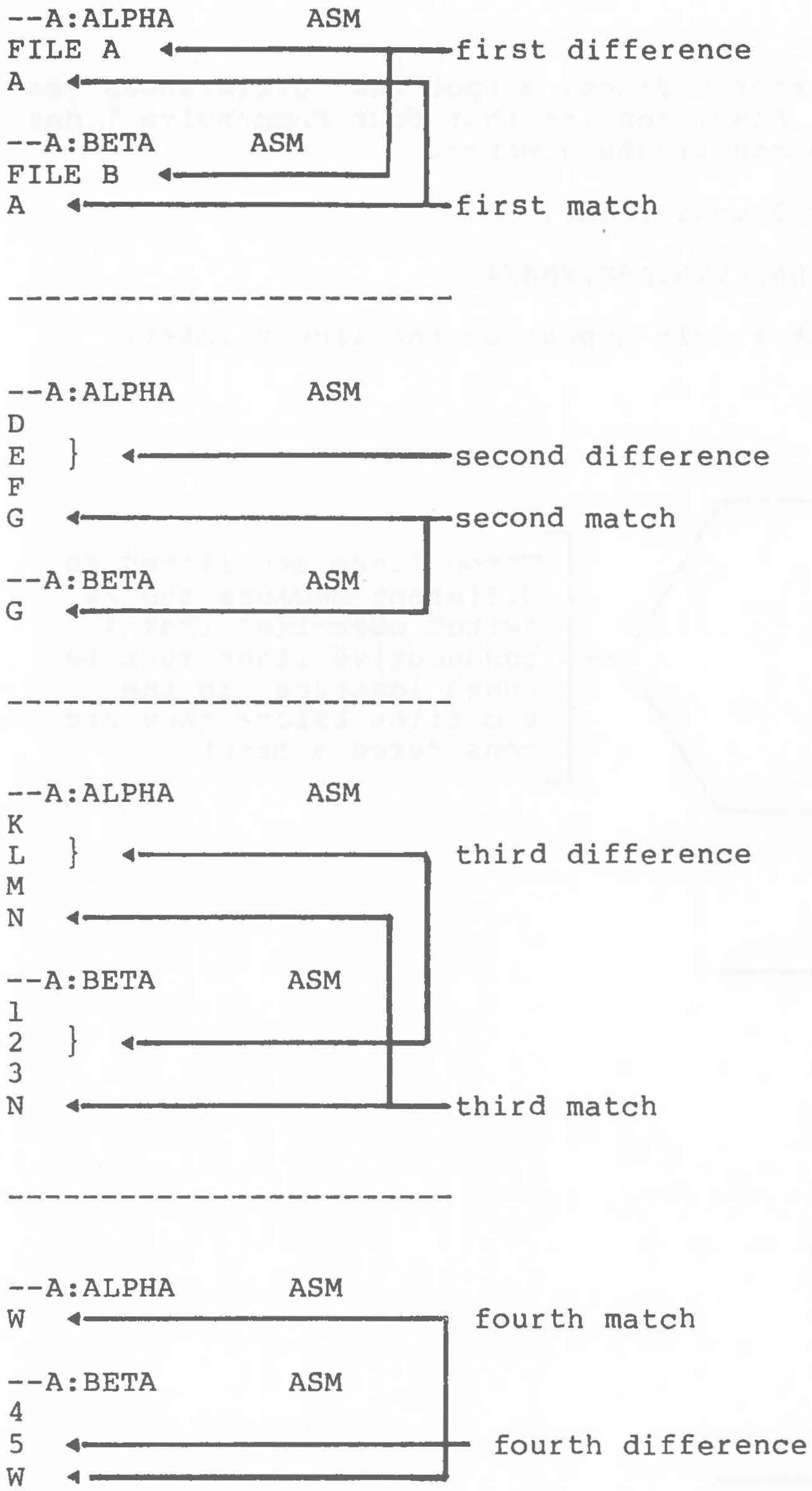
To compare the two files and output the differences on the terminal screen, enter the following (method 2) command line:

```
         FILCOM ALPHA,BETA.ASM,CON
```

FILCOM is directed to compare ALPHA.ASM with BETA.ASM and output the differences on the terminal screen. All other defaults remain intact (do not use tabs, spaces, or comments for matches, and conduct a source comparison).

The output appears as follows on the terminal screen:

```
--A:ALPHA        ASM
FILE A  <-----------------------first difference
A  <----------------------------
                                
--A:BETA      ASM
FILE B  <-----------------------
A  <----------------------------first match
```



```
--A:ALPHA        ASM
FILE A
A

--A:BETA      ASM
FILE B
A

------------------------

--A:ALPHA        ASM
D
E  }
F
G

--A:BETA         ASM
G

------------------------

--A:ALPHA        ASM
K
L  }
M
N

--A:BETA        ASM
1
2  }
3
N

------------------------

--A:ALPHA       ASM
W

--A:BETA        ASM
4
5
W
```

Example 2

Using the same two source files, output the differences on the line printer. Also, require that four successive lines must be the same to constitute a match.

Using invoke method 2 again, enter:

```
FILCOM ALPHA,BETA.ASM,PRN/4
```

The following output should appear on the line printer:

```
--A:ALPHA       ASM
FILE A
A
B
C
D
E
F
G

--A:BETA        ASM
FILE B
A
B
C
G
```

These lines are listed as different because the /4 switch specifies that 4 consecutive lines must be found identical in the two files before they are considered a match.

```
--A:ALPHA       ASM
K
L
M
N

--A:BETA        ASM
1
2
3
N
```

```
--A:ALPHA       ASM
W

--A:BETA        ASM
4
5
W
```

Example 3

Using the same two source files again, force a binary comparison, then output the differences on the terminal screen.

Using invoke method 1,enter:

```
        FILCOM
```

FILCOM responds:

```
        Source 1 Filename [.ASM]:
```

Entries and responses should appear as follows:

```
        Source 1 Filename [.ASM]:  ALPHA/B
        Source 2 Filename [ALPHA.BAK]:  BETA.ASM
        List Filename [ALPHA.DIF]:  CON
```

The /B switch at the end of the Source 1 line forces binary comparison. This switch, and the others, may be entered at the end of any of the response entries. The switches may be, but need not be, entered on the same response line.

The screen display should appear as follows:

```
ADDRESS ALPHA ASM      BETA ASM
 00006   41       42
 00012   44       47
 00015   45       48
 00018   46       49
 0001B   47       4A
 0001E   48       31
 00021   49       32
 00024   4A       33
 00027   4B       4E
 0002A   4C       4F
 0002D   4D       50
 00030   4E       51
 00033   4F       52
 00036   50       53
 00039   51       54
 0003C   52       55
 0003F   53       56
 00042   54       34
 00045   55       35
 00048   56       57
 0004B   57       58
 0004E   58       59
 00051   59       5A
```

APPENDIX A

Instructions for Single Disk Drive Users

For single disk drive users the commands are exactly the same syntax as for two drive users. The difference lies in your perception of the "arrangement" of the drives.

You must think of his system as having two disk drives: drive A: and drive B:. However, instead of A: and B: designating physical disk drive mechanisms, the A: and B: designate diskettes. Therefore, when the user specifies drive B: while operating on drive A: (the prompt is A:), MS-DOS prompts the user to "switch drives" by swapping diskettes.

The prompts are:

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

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

These procedures apply to any MS-DOS COMMAND commands (both internal and external) that can request or direct a different drive as a part of its syntax. These commands include:

```
CHKDSK [d:]
COPY <filespec>[filespec]
DEL <filespec> [filespec...]
DIR [d:][filename]
FORMAT d:[/S]
RENAME <filespec> <filename>
TYPE <filespec>
```

Also, if any of these commands are used in a BATCH file and call for a different drive, the single disk drive procedures apply. Execution is halted and the appropriate prompt is displayed.

EXAMPLE

The following example may serve as an illustration for all of the commands listed above:

```
A:COPY COMMAND.COM B:  Insert  diskette  for  drive
B: and depress space bar when ready
    1 File(s) copied

Insert diskette for drive A: and depress space  bar
when ready
A:_
```

APPENDIX B

MS-DOS File Control Block Definition

The MS-DOS File Control Block (FCB) is defined as follows:

byte 0 Drive Code. Zero specifies the default drive, 1=drive A, 2=drive B, etc.

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

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

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

bytes 14-15 Size of the record the user wishes to work with. This word may be filled immediately after an OPEN of the file if the default logical record size(128 bytes) is not desired. The Open and Create functions set this field to 128; it is also changed to 128 if a read or write is attempted with the field set to zero.

bytes 16-19 File size. This is the current size, in bytes, of the file. It may read by user programs but must not be written by them.

bytes 20-21 Date. This is normally the date of the last write to the file. It is set by all disk write operations and Create to today's date. It is set by Open to the date recorded in the disk directory for the file. User programs may modify this field after writing to a file but before closing it to change the date recorded in the disk directory.

The format of this 16-bit field is as follows: bits 0-4, day of month; bits 5-8, month of year; bits 9-15, year minus 1980. All zero means no date.

bytes 22-23 Time. Similar to Date, above. The format is bits 0-4, seconds/2; bits 5-10, minutes; bits 11-15, hours.

bytes 24-31 Reserved for MS-DOS.

byte 32 Current record. Identifies the record within the current block of 128 records that will be accessed with a sequential read or write function. See Current Block,bytes 12-13.

bytes 33-36 Random Record. This field must only be set if the file is to be accessed with a random read or write function. If the record size is greater than or equal to 64 bytes, only the first 3 bytes are used, as a 24-bit number representing the position in the file of a record. If the record size if less than 64 bytes, all 4 bytes are used as 32-bit number of the same purpose. This field is thus large enough to address any byte in a file of the maximum size, 230 bytes.

THE EXTENDED FCB

The extended FCB is a special format used to search for files in the disk directory with special attributes. It consists of 7 bytes in front of a normal FCB, formatted as follows:

FCB-7 Flag. FF hex is placed here to signal an extended FCB.

FCB-6 - FCB-2 Zero field

FCB-1 Attribute byte. If bit 1 = 1, hidden files will be included in directory searches. If bit 2 = 1, system files will be included in directory searches.

Any reference in the description of MS-DOS function calls to an FCB, whether opened or unopened, may use either a normal FCB or an extended FCB. A normal FCB has the same effect as an extended FCB with the attribute byte set to zero.

APPENDIX C

MS-DOS INTERRUPTS AND FUNCTION CALLS

C.1 INTERRUPTS

MS-DOS Reserves interrupt types 20 to 3F hex for its use. This means absolute locations 80 to FF hex are the transfer address storage locations reserved by the DOS. The defined interrupts are as follows with all values in hex:

20 Program terminate. This is the normal way to exit a program. This vector transfers to the logic in the DOS for restoration of <CONTROL-C> exit addresses to the values they had on entry to the program. All file buffers are flushed to disk. All files that have changed in length should have been closed (see function call 10 hex) prior to issuing this interrupt. If the changed file was not closed its length will not be recorded correctly in the directory. When this interrupt is executed, CS MUST point to the 100H parameter area.

21 Function request. See section II FUNCTION REQUESTS.

22 Terminate address. The address represented by this interrupt(88-8B hex) is the the address to which control will transfer when the program terminates. This address is copied into low memory of the segment the program is loaded into at the time this segment is created. If a program wishes to execute a second program it must set the terminate address prior to creation of the segment the program will be loaded into. Otherwise once the second program executes its termination would cause transfer to its host's termination address.

23 <CONTROL-C> exit address. If the user types <CONTROL-C> during keyboard input or video output, "^C" will be printed on the console and an interrupt type 23 hex will be executed. If the <CONTROL-C>

routine preserves all registers, it may end with a a return-from-interrupt instruction (IRET) to continue program execution. If functions 9 or 10 (buffered output and input), were being executed, then I/O will continue from the start of the line. When the interrupt occurs, all registers are set to the value they had when the original call to MS-DOS was made. There are no restrictions on what the <CONTROL-C> handler is allowed to do, including MS-DOS function calls, so long as the registers are unchanged if IRET is used.

If the program creates a new segment, loads in a second program which itself changes the <CONTROL-C> address, the termination of the second program and return to the first will cause the <CONTROL-C> address to be restored to the value it had before execution of the second program.

24 Fatal error abort vector. When a fatal error occurs within MS-DOS, control will be transferred with an INT 24H. On entry to the error handler, AH will have its bit 7=0 if the error was a hard disk error (probably the most common occurrence), bit 7=1 if not. If it is a hard disk error, bits 0-2 include the following:

bit 0   0 if read, 1 if write

bit 2	1	AFFECTED DISK AREA
0	0	Reserved area
0	1	File allocation table
1	0	Directory
1	1	Data area

AL, CX, DX, and DS:BX will be setup to perform a retry of the transfer with INT 25H or INT 26H (below). DI will have a 16-bit error code returned by the hardware.

The values returned are:

0	write protect
2	disk not ready
4	data error
6	Seek error
8	Sector not found
A	Write fault
C	General disk failure

The registers will be set up for a BIOS disk call and the returned code will be in the lower half of the DI register with the upper half undefined. The user stack will look as follows from top to bottom:

```
IP     Registers such that if an IRET is executed
CS     the DOS will respond according to (AL)
FLAGS  as follows:

       (AL)=0    ignore the error
           =1    retry the operation
                 (IF THIS OPTION USED STACK DS,
                  BX,CX AND DX MUST NOT BE MODIFIED!)
           =2 abort the program

AX     USER REGISTERS AT TIME OF REQUEST
BX
CX
DX
SI
DI
BP
DS
ES
IP     The interrupt from the user to the DOS
CS
FLAGS
```

Currently, the only error possible when AH bit 7=1 is a bad memory image of the file allocation table.

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 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 (0=A, 1=B, etc.)
(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 boundarys. For example logical record number 0 is track 0 sector 1, whereas logical record number 12 hex is track 2 sector 3.

All registers but 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:

Return	Description
0	Attempt to write on write protected disk
2	Disk not ready
4	Data error
6	Seek error
8	Sector not found
C	General disk failure
A	Write fault

26 Absolute disk write. This vector is the counterpart to 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 which are to remain resident when COMMAND regains control. Such a program is loaded as an executing COM file by COMMAND. After it has initialized itself, it must set DX to its last address plus one in the segment it is executing in, then execute an interrupt 27H. COMMAND will then treat the program as an extension of MS-DOS, and the program will not be overlaid when other programs are executed.

C.2 FUNCTION REQUESTS

The user requests a function 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 21 hex. When MS-DOS takes control it switches to an internal stack. User registers except AX 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 80 hex in addition to the user needs. There is an additional mechanism provided for programs that conforms to CP/M calling conventions. The function number is placed in the CL register, other registers are set as normal according to the function specification, and an intrasegment call is made to location 5 in the current code segment. This method is only available to functions which do not pass a parameter in AL and whose numbers are equal to or less than 36. Register AX is always destroyed if this mechanism is used, otherwise it is the same as normal function requests. The functions are as follows with all values in hex:

0 Program terminate. The terminate and <CONTROL-C> exit addresses are restored to the values they had on entry to the terminating program. All file

buffers are flushed, but files which have been changed in length but not closed will not be recorded properly in the disk directory. Control transfers to the terminate address.

1 Keyboard input. Waits for a character to be typed at the keyboard, then echos the character to the video device and returns it in AL. The character is checked for a <CONTROL-C>. If this key is detected an interrupt 23 hex will be executed.

2 Video output. The character in DL is output to the video device. If a <CONTROL-C> is detected after the output an interrupt 23 hex will be executed.

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

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

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

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

7 Direct console input. Waits for a character to be typed at the keyboard, 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 "$" (24 hex). Each character in the string will be output to the video device in the same form as function 2.

A Buffered keyboard input. On entry, DS:DX point to an input buffer. The first byte must not be zero and specifies the number of characters the buffer can hold. Characters are read from the keyboard and placed in the buffer beginning at the third byte. Reading the keyboard and filling the buffer continues until <RETURN> is typed. If the buffer fills to one less than the maximum, then additional keyboard input is ignored until a <RETURN> is typed. The second byte of the buffer is set to the number of characters received excluding the carriage return (0D hex), which is always the last character. Editing of this buffer is described in the main MS-DOS document under "template editing".

B Check keyboard status. If a character is available from the keyboard, AL will be FF hex, Otherwise AL will be 00.

C Character input with buffer flush. First the keyboard type-ahead buffer is emptied. Then if AL is 1, 6, 7, 8, or 0A hex, the corresponding MS-DOS input function is executed. If AL is not one of these values, no further operation is done and AL returns 00.

D Disk reset. Flushes all file buffers. Unclosed files that have been changed in size will not be properly recorded in the disk directory until they are closed. This function need not be called before a disk change if all files which have been written have been closed.

E Select disk. The drive specified in DL (0=A, 1=B, etc) is selected as the default disk. The number of drives is returned in AL.

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

If the drive code was 0 (default disk), it is changed to actual disk used (A=1, B=2, etc.) This allows changing the default disk without interfering with subsequent operations on this file. The high byte of the current block field is set to zero. The size of the record to be worked with(FCB bytes E-F hex) is set to the system default of 80 hex. The size of the file, and the time and date are set in the FCB from information obtained from the directory.

It is the user's responsibility to set the record size (FCB bytes E-F) to the size he wishes to think of the file in terms of, if the default 80 hex is not appropriate. It is also the user's responsibility to set the random record field and/or current block and record fields.

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 the directory, it is assumed the disk has been changed and AL returns FF hex. 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 name (name could have "?"'s indicating any letter matched) and if none are found AL returns FF hex. Otherwise, locations at the disk transfer address are set as follows:

1. If the FCB provided for searching was an extended FCB, then the first byte is set to FF hex, then 5 bytes of zeros, then the attribute byte from the search FCB, then the drive number used (A=1, B=2, etc.), 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.

2. If the FCB provided for searching was a normal FCB, then the first byte is set to the drive number used (A=1, B=2, etc.) and the next 32 bytes contain the matching directory entry. Thus the disk transfer address contains a valid unopened normal FCB.

Directory entries are formatted as follows:

Location	Bytes	Description
0	11	File name and extension
11	1	Attributes. Bits 1 or 2 make file hidden
10	10	Zero field (for expansion)
22	2	Time. Bits 0-4 = secs*2 5-10 = min 11-15 = hrs
24	2	Date. Bits 0-4 = day 5-8 = month 9-15 = year
26	2	First allocation unit
28	4	File size, in bytes. (30 bits max.)

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 are the same as function 11. The reserved area of the FCB keeps information necessary for continuing the search, so it must not be modified.

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 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. 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 room in the disk transfer segment to read one record, so the transfer was aborted. 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 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 disk is full AL returns with a 01. A return of 02 means there was not enough room in the disk transfer segment to write one record, so the transfer was aborted. 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 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.

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+11 hex) in what is normally a reserved area. Every matching occurrence of the first is changed to the second(with the restriction that two files cannot have the exact 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 hex if no match was found, otherwise 00.

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

1A Set disk transfer address. The disk transfer address is set to DS:DX. MS-DOS will not allow disk transfers to wrap around within the segment, nor to overflow into the next segment.

1B Allocation table address. On return, DS:BX point to the allocation table for the current drive, DX has the number of allocation units, and AL has the number of records per allocation unit, and CX has the size of the physical sector. At DS:[BX-1], the byte before the allocation table, is the dirty byte for the table. If set to 01, it means the table has been modified and must be written back to disk. If 00, the table is not modified. Any programs which get the address and directly modify the table must be sure to set this byte to 01 for the changes to be recorded. This byte should NEVER be set to 00 - instead, a DISK RESET function (#0D hex) should be performed to write the table and reset the bit.

21 Random read. On entry, DS:DX point to an opened FCB The current block and current record are set to agree with the random record field, then the record addressed by these fields is loaded 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 room in the disk transfer segment to read one record, so the transfer was aborted. 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 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 disk is full AL returns 01. A return of 02 means there was not enough room in the disk transfer segment to write one record, so the transfer was aborted. AL returns 00 if the transfer was completed successfully.

23 File size. On entry, DS:DX point to an unopened FCB. The disk directory is searched for the first matching entry and if none is found, AL returns FF. Otherwise the random record field is set with the size of the file(in terms of the record size field rounded up) and AL returns 00.

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 vector. The interrupt type specified in AL is set to the 4-byte address 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 100 hex 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 is updated and the current termination and <CONTROL-C> exit addresses are saved in the new program segment starting at 0A hex

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 03 indicates the last record is a partial record. If wrap-around above address FFFF hex in the disk transfer segment would occur, as many records as possible are read and AL returns 02. If all records are read successfully, AL returns 00. In any case CX returns with the actual number of records read, and the random record field and the current block/record fields are set to address the next record.

28 Random block write. Essentially the same as function 27 above, except for writing and a write-protect indication. 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 file name. On entry DS:SI points to a command line to parse, and ES:DI points to a portion of memory to be filled in with an unopened FCB. Leading TABs and spaces are ignored when scanning. If bit 0 of AL is equal to 1 on entry, then at most one leading file name separator will be ignored, along with any trailing TABs and spaces. The four filename separators are:

: ; , = +

If bit 0 of AL is equal to 1, then all parsing stops if a separator is encountered. The command line is parsed for a file name of the form d:filename.ext, and if found, a corresponding unopened FCB is created at ES:DI. The entry value of AL bits 1, 2, and 3 determine what to do if the drive, filename, or extension, respectively, are missing. In each care, if the bit is a zero and the field is not present on the command line, then the FCB is filled with a fixed value (0, meaning the default drive for the drive field ; all blanks for the filename and extension fields). If the bit is a 1, and the field is not present on the command line, then that field in the destination FCB at ES:DI is left unchanged. If an asterisk "*" appears in the filename or extension, then all remaining characters in the name or extension are set to "?".

The following characters are illegal within MS-DOS file specifications:

The following characters are illegal within MS-DOS file specifications:

"/ [ ] + = ; ,

Control characters and spaces also may not be given as elements of file specifications. If any of these characters are encountered while parsing, or the period (.) or colon (:) is found in an invalid position, then parsing stops at that point.

If either "?" or "*" appears in the file name or extension, then AL returns 01, otherwise 00. DS:SI will return pointing to the first character after the file name.

2A Get date. Returns date in CX:DX. CX has the year, 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 will be 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, then AL returns 00. If the date is not valid, then 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 easily converted to a printable form yet can also be calculated upon (e.g., subtracting two times).

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.

2E Set/Reset Verify Flag. On entry, DL must be 0 and AL has the verify flag: 0 = no verify, 1 = verfiy after write. this flag is simply passed to the I/O system on each write, so sts exact meaning is interpreted there.

APPENDIX D

Disk Errors

If a disk error occurs at any time during any command or program, MS-DOS retries the operation three times. If the operation cannot be completed successfully, MS-DOS returns an error message in the following format:

```
<type> ERROR WHILE <I/O action> ON DRIVE x
Abort,Ignore,Retry:_
```

In this message,<type> may be one of the following:

```
WRITE PROTECT
NOT READY
SEEK
DATA
SECTOR NOT FOUND
WRITE FAULT
DISK
```

The <I/O-action> may be either of the following:

```
READING
WRITING
```

The drive <d> indicates the drive in which the error
has occurred.
MS-DOS waits entry of one of the following responses:

A Abort. Terminate the program requesting the disk read or write.

I Ignore. Ignore the bad sector and pretend the error did not occur.

R Retry. Repeat the operation. This response is particularly useful if the operator has corrected the error (such as with NOT READY or WRITE PROTECT).

Usually, you will want to attempt recovery by entering responses in the order:

R (to try again)
A (to terminate program and try a new disk)

One other error message might be related to faulty disk read or write:

FILE ALLOCATION TABLE BAD FOR DRIVE x

This message means that the copy in memory of one of the allocation tables has pointers to nonexistent blocks. Possibly the disk was not FORMATted before use.

INDEX

utility
software
package
reference manual

for 8086 microprocessors

Microsoft, Inc.
Microsoft Building
10700 Northup Way
Bellevue, WA 98004

LIMITED WARRANTY

MICROSOFT, Inc. shall have no liability or responsibility to purchaser or to any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this product. This product will be exchanged within twelve months from date of purchase if defective in manufacture, labeling, or packaging, but except for such replacement the sale or subsequent use of this program is without warranty or liability.

THE ABOVE IS A LIMITED WARRANTY AND THE ONLY WARRANTY MADE BY MICROSOFT, INC. ANY AND ALL WARRANTIES FOR MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY EXCLUDED.

To report software bugs or errors in the documentation, please complete and return the Problem Report at the back of this manual.

8407-100
Part Number 14F06A

Package Contents

1 diskette, with the following files:
- MASM.EXE
- LINK.EXE
- LIB.EXE
- CREF.EXE

1 binder with 4 manuals
- MS-LINK Linker Utility Manual
- MS-LIB Library Manager Manual
- MS-CREF Cross Reference Facility Manual

System Requirements

Each utility requires different amounts of memory.

MS-LINK - 54K bytes of memory minimum:
44K bytes for code
10K bytes for run space

MS-LIB - 38K bytes of memory minimum:
28K bytes for code
10K bytes for run space

MS-CREF - 24K bytes of memory minimum:
14K bytes for code
10K bytes for run space

1 disk drive

1 disk drive if and only if output is sent to the same physical diskette from which the input was taken. None of the utility programs in this package allow time to swap diskettes during operation on a one-drive configuration. Therefore, two disk drives is a more practical configuration.

Microsoft

Welcome to the Microsoft family of products.

Microsoft, Inc. is recognized as the leader in microcomputer software. Microsoft BASIC interpreter, in its several versions, has become the standard high-level programming language used in microcomputers. Microsoft, Inc. continues to supply consistently high-quality software which sets the standard for software quality for all types of users.

In addition to the Utility Software Package and Microsoft BASIC interpreter, Microsoft sells other full-feature language compilers, language subsets, and operating system products. Microsoft offers a "family" of software products that both look alike from one product to the next, and can be used together for effective program development.

For more information about other Microsoft products, contact:

Microsoft, Inc.
10700 Northup Way
Bellevue, WA 98004
(206) 828-8080

Contents

MS-LIB Library Manager

MS-CREF Cross Reference Facility

GENERAL INTRODUCTION

The Microsoft Utility Software Package includes four utility programs used for developing assembly language programs. In addition, the MS-LINK Linker Utility is used with all of Microsoft's 16-bit language compilers.

Major Features

MS-LINK Linker Utility

MS-LINK is a virtual linker, which can link programs that are larger than available memory

MS-LINK produces relocatable executable object code.

MS-LINK knows how to handle user-defined overlays.

MS-LINK can perform multiple library searches, using a dictionary library search method.

MS-LINK prompts the user for input and output modules and other link session parameters.

MS-LINK can be run with an automatic response file to answer the linker prompts.

MS-LIB Library Manager

MS-LIB can add, delete, and extract modules in the user's library of program files.

MS-LIB prompts the user for input and output file and module names.

MS-LIB can be run with an automatic response file to answer the library prompts.

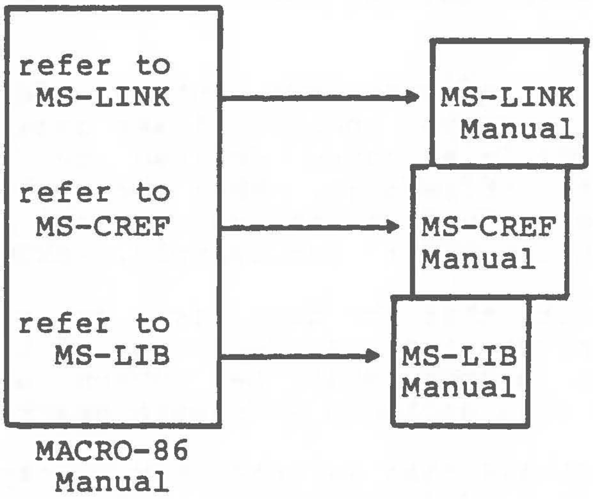
MS-LIB produces a cross reference of symbols in the library modules.

MS-CREF Cross Reference Facility

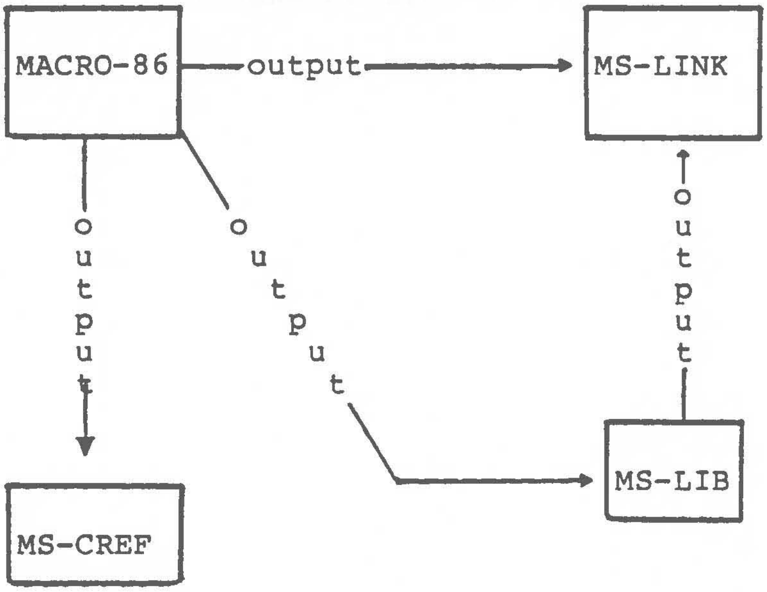
MS-CREF produces a cross reference listing of all symbolic names in the source program, giving both the source line number of the definition and the source line numbers of all other references to them.

Using These Manuals

These manuals are designed to be used as a set and individually. Each manual is mostly self-contained and refers to the other manuals only at junctures in the software. The Overview given below describes generally the flow of program development from creating a source file through program execution. The processes described in this overview are echoed and expanded in overviews in each of the three manuals.



Each of the three manuals is used independently. References between manuals reflect junctures in the software.



Syntax Notation

The following notation is used throughout this manual in descriptions of command and statement syntax:

[ ] Square brackets indicate that the enclosed entry is optional.

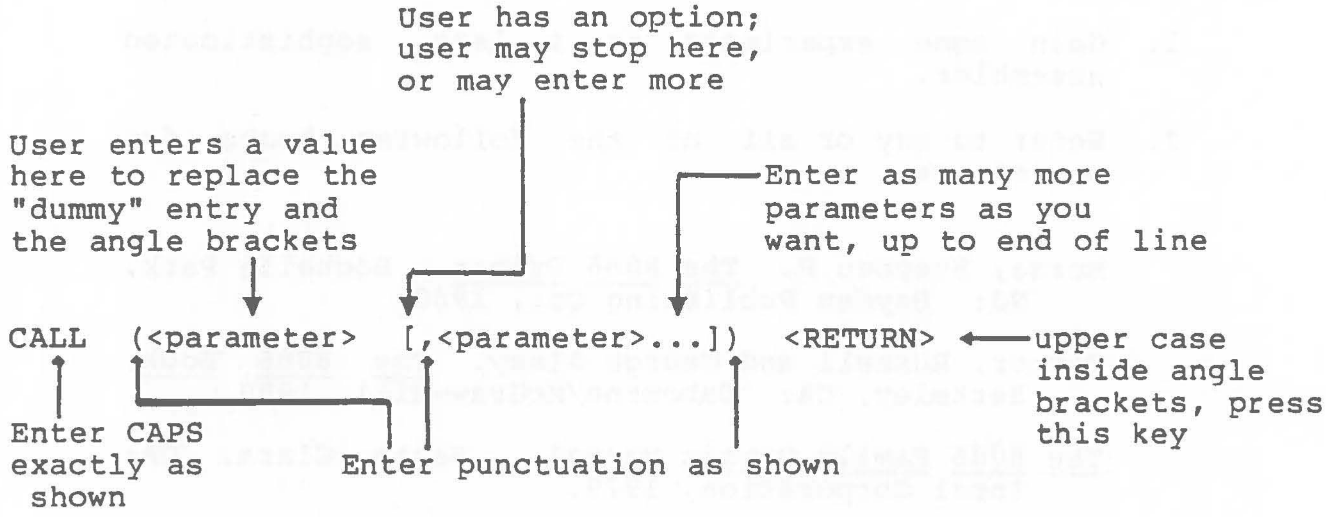
< > Angle brackets indicate user entered data. When the angle brackets enclose lower case text, the user must type in an entry defined by the text; for example, <filename>. When the angle brackets enclose upper case text, the user must press the key named by the text; for example, <RETURN>.

{ } Braces indicate that the user has a choice between two or more entries. At least one of the entries enclosed in braces must be chosen unless the entries are also enclosed in square brackets.

... Ellipses indicate that an entry may be repeated as many times as needed or desired.

CAPS Capital letters indicate portions of statements or commands that must be entered, exactly as shown.

All other punctuation, such as commas, colons, slash marks, and equal signs, must be entered exactly as shown.


User has an option;
user may stop here,
or may enter more
User enters a value
here to replace the
"dummy" entry and
the angle brackets
Enter as many more
parameters as you
want, up to end of line
CALL (<parameter> [,<parameter>...]) <RETURN>
upper case
inside angle
brackets, press
this key
Enter CAPS
exactly as
shown
Enter punctuation as shown

Learning More About Assembly Language Programming

These manuals explain how to use Microsoft's Utility Software Package, but they do not teach users how to program in assembly language.

We assume that the user of The Utility Software Package will have had some experience programming in assembly language. If you do not have any experience, we suggest two courses:

1. Gain some experience on a less sophisticated assembler.

2. Refer to any or all of the following books for assistance:

   Morse, Stephen P. The 8086 Primer. Rochelle Park, NJ: Hayden Publishing Co., 1980.

   Rector, Russell and George Alexy. The 8086 Book. Berkeley, CA: Osbourne/McGraw-Hill, 1980.

   The 8086 Family User's Manual. Santa Clara, CA: Intel Corporation, 1979.

   8086/8087/8088 Macro Assembly Language Reference Manual. Santa Clara, CA: Intel Corporation, 1980.

NOTE

Some of the information in these books was based on preliminary data and may not reflect the final functional state. Information in your Microsoft manuals was based on Microsoft's development of its 16-bit software for the 8086 and 8088.

Overview of Program Development

This overview describes generally the steps of program development. Each step is described fully in the individual product manuals. The numbers in the descriptions match the numbers in the facing diagram.

1. Use EDLIN (the editor in Microsoft's MS-DOS), or other 8086 editor compatible with your operating system, to create an 8086 assembly language source file. Give the source file the filename extension .ASM (ASMRO-86 recognizes .ASM as default).

2. Assemble the source file with MACRO-86, which outputs an assembled object file with the default filename extension .OBJ (2a). Assembled files, the user's program files (2b), can be linked together in step 3.

   MACRO-86 (optionally) creates two types of listing file:

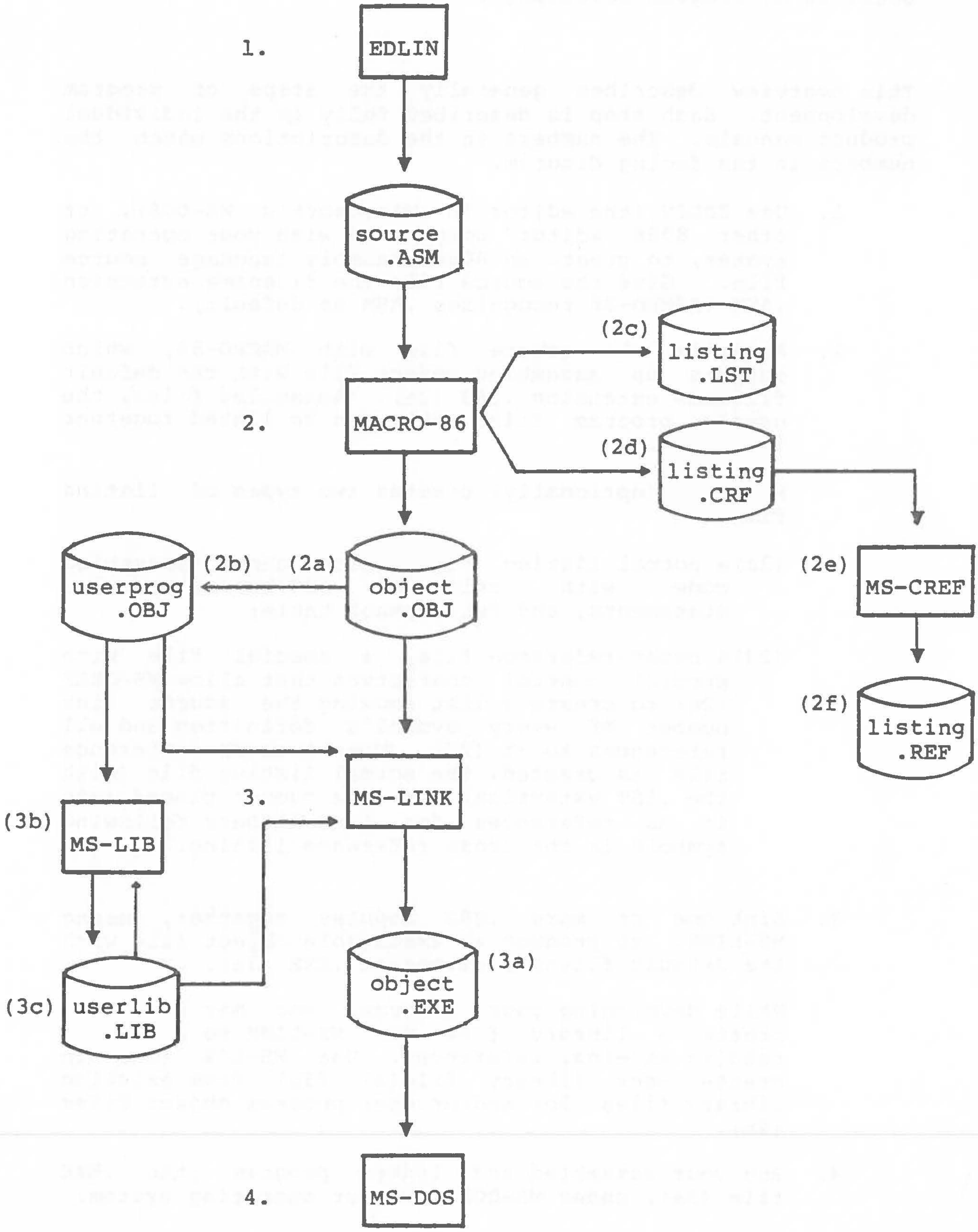
   (2c) a normal listing file which shows assembled code with relative addresses, source statements, and full symbol table;

   (2d) a cross-reference file, a special file with special control characters that allow MS-CREF (2e) to create a list showing the source line number of every symbol's definition and all references to it (2f). When a cross reference file is created, the normal listing file (with the .LST extension) has line number placed into it as references for line numbers following symbols in the cross reference listing.

3. Link one or more .OBJ modules together, using MS-LINK, to produce an executable object file with the default filename extension .EXE (3a).

   While developing your program, you may want to create a library file for MS-LINK to search to resolve external references. Use MS-LIB (3b) to create user library file(s) (3c) from existing library files (3c) and/or user program object files (2b).

4. Run your assembled and linked program ,the .EXE file (3a), under MS-DOS, or your operating system.


1.
EDLIN
source
.ASM
(2c)
listing
.LST
2.
MACRO-86
(2d)
listing
.CRF
(2b)
(2a)
userprog
.OBJ
object
.OBJ
(2e)
MS-CREF
(2f)
listing
.REF
3.
MS-LINK
(3b)
MS-LIB
(3a)
object
.EXE
(3c)
userlib
.LIB
4.
MS-DOS

MS-LINK
linker
utility

Microsoft, Inc.
Microsoft Building
10700 Northup Way
Bellevue, WA 98004

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft, Inc. The software described in this document is furnished under a license agreement or non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy The MS-LINK Linker Utility on cassette tape, disk, or any other medium for any purpose other than personal convenience.

LIMITED WARRANTY

MICROSOFT, Inc. shall have no liability or responsibility to purchaser or to any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this product. This product will be exchanged within twelve months from date of purchase if defective in manufacture, labeling, or packaging, but except for such replacement the sale or subsequent use of this program is without warranty or liability.

THE ABOVE IS A LIMITED WARRANTY AND THE ONLY WARRANTY MADE BY MICROSOFT. ANY AND ALL WARRANTIES FOR MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY EXCLUDED.

To report software bugs or errors in the documentation, please complete and return the Problem Report at the back of this manual.

8407B-100-01

System Requirements

The MS-LINK Linker Utility requires:

49K bytes of memory minimum:
- 40K bytes for code and data
- 10K bytes for run space

1 disk drive

1 disk drive if and only if output is sent to the same physical diskette from which the input was taken. MS-LINK does not allow time to swap diskettes during operation on a one-drive configuration. Therefore, two disk drives is a more practical configuration.

Contents

INTRODUCTION

Features and Benefits of MS-LINK

MS-LINK is a relocatable linker designed to link together separately produced modules of 8086 object code. The object modules must be 8086 files only.

MS-LINK is user-friendly. For all the necessary and optional commands, MS-LINK prompts the user. The user's answers to the prompts are the commands for MS-LINK.

The output file from MS-LINK (Run file) is not bound to specific memory addresses and, therefore, can be loaded and executed at any convenient address by the user's operating system.

MS-LINK uses a dictionary-indexed library search method, which substantially reduces link time for sessions involving library searches.

MS-LINK is capable of linking files totaling 384K bytes.

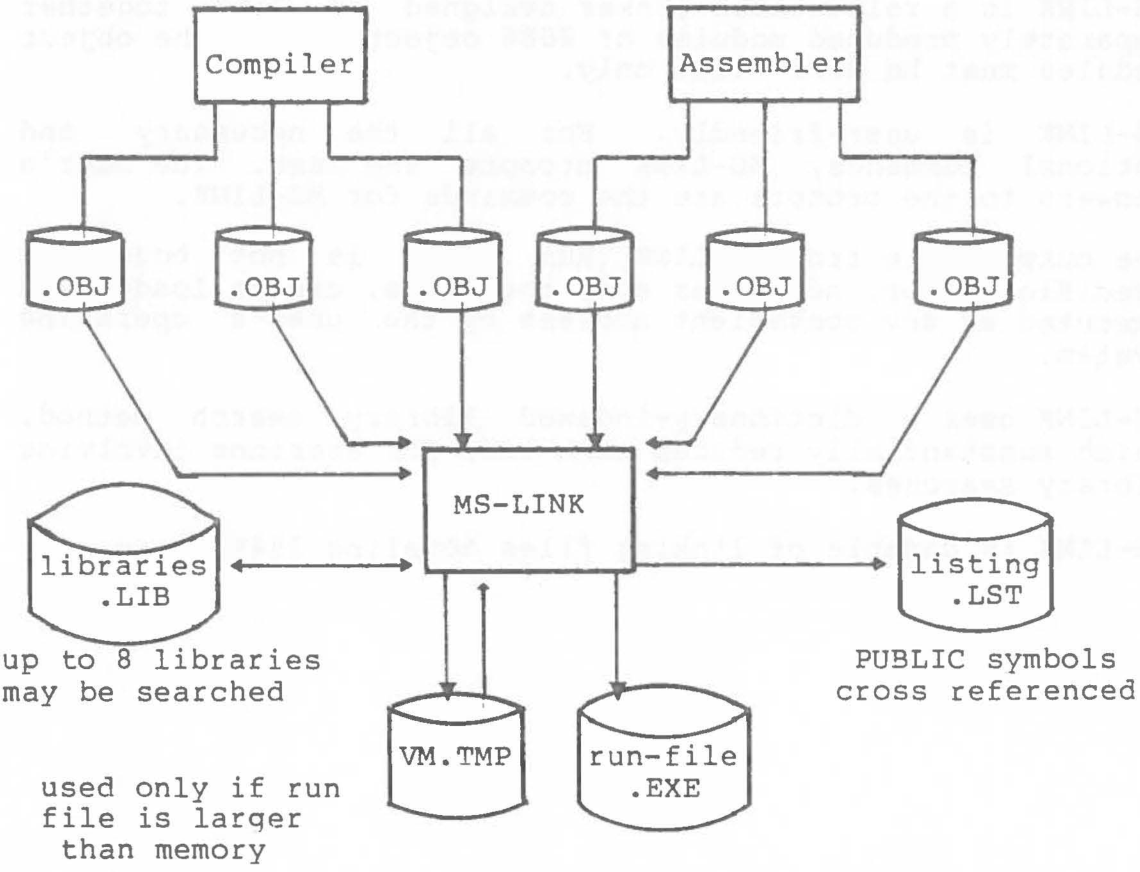
Overview of MS-LINK Operation

MS-LINK combines several object modules into one relocatable load module, or Run file.

As it combines modules, MS-LINK resolves external references between object modules and can search multiple library files for definitions for any external references left unresolved.

MS-LINK also produces a list file that shows external references resolved and any error messages.

MS-LINK uses available memory as much as possible. When available memory is exhausted, MS-LINK then creates a disk file and becomes a virtual linker.



Definitions

Three terms will appear in some of the error messages listed in Chapter 2. These terms describe the underlying functioning of MS-LINK. An understanding of the concepts that define these terms provides a basic understanding of the way MS-LINK works.

1. Segment

   A Segment is a contiguous area of memory up to 64K bytes in length. A Segment may be located anywhere in 8086 memory on a "paragraph" (16 byte) boundary. The contents of a Segment are addressed by a Segment-register/offset pair.

2. Group

   A Group is a collection of Segments which fit within 64K bytes of memory. The Segments are named to the Group by the assembler, by the compiler, or by you. The Group name is given by you in the assembly language program. For the high-level languages (BASIC, FORTRAN, COBOL, Pascal), the naming is carried out by the compiler.

   The Group is used for addressing Segments in memory. Each Group is addressed by a single Segment register. The Segments within the Group are addressed by the Segment register plus an offset. MS-LINK checks to see that the object modules of a Group meet the 64K byte constraint.

3. Class

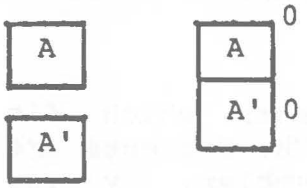
   A Class is a collection of Segments. The naming of Segments to a Class controls the order and relative placement of Segments in memory. The Class name is given by you in the assembly language program. For the high-level languages (BASIC, FORTRAN, COBOL, Pascal), the naming is carried out by the compiler. The Segments are named to a Class at compile time or assembly time. The Segments of a Class are loaded into memory contiguously. The Segments are ordered within a Class in the order MS-LINK 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 MS-LINK. Classes may be loaded across 64K byte boundaries. The Classes will be divided into Groups for addressing.

How MS-LINK Combines and Arranges Segments

MS-LINK works with four combine types, which are declared in the source module for the assembler or compiler: private, public, stack, and common. (The memory combine type available in Microsoft's MACRO-86 is treated the same as public. MS-LINK does not automatically place memory combine type as the highest segments.)

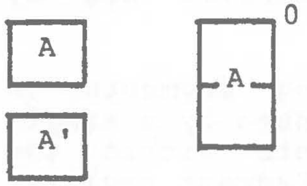
MS-LINK combines segments for these combine types as follows:

Private



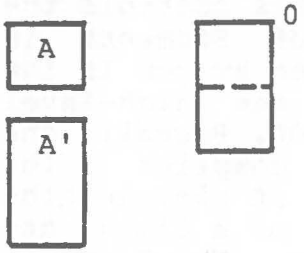
Private segments are loaded separately and remain separate. They may be physically contiguous but not logically, even if the segments have the same name. Each private segment has its own base address.

Public



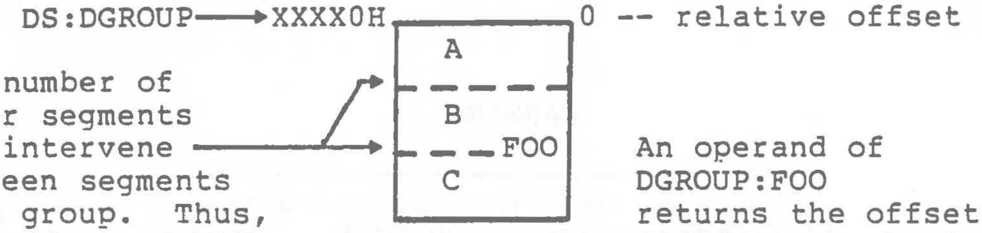
Public segments of the same name and class name are loaded contiguously. Offset is from beginning of first segment loaded through last segment loaded. There is only one base address for all public segments of the same name and class name. (Combine types stack and memory are treated the same as public. However, the Stack Pointer is set to the first address of the first stack segment.)

Common



Common segments of the same name and class name are loaded overlapping one another. There is only one base address for all common segments of the same name. The length of the common area is the length of the longest segment.

Placing segments in a Group in the assembler provides offset addressing of items from a single base address for all segments in that Group.



Any number of other segments may intervene between segments of a group. Thus, the offset of FOO may. be greater than the size of segments in group combined, but no larger than 64K.

An operand of DGROUP:FOO returns the offset of FOO from the beginning of the first segment of DGROUP (segment A here)

Segments are grouped by declared class names. MS-LINK loads all the segments belonging to the first class name encountered, then loads all the segments of the next class name encountered, and so on until all classes have been loaded.

```
If your program contains:            They will be loaded as:

   A  SEGMENT 'FOO'                      'FOO'
   B  SEGMENT 'BAZ'                        A
   C  SEGMENT 'BAZ'                        E
   D  SEGMENT 'ZOO'                      'BAZ'
   E  SEGMENT 'FOO'                        B
                                           C
                                         'ZOO'
                                           D
```

If you are writing assembly language programs, you can exercise control over the ordering of classes in memory by writing a dummy module and listing it first after the MS-LINK Object Modules prompt. The dummy module declares segments into classes in the order you want the classes loaded.

WARNING

> Do not use this method with BASIC, COBOL, FORTRAN, or Pascal programs. Allow the compiler and the linker to perform their tasks in the normal way.

For example:

```
        A  SEGMENT  'CODE'
        A  ENDS
        B  SEGMENT  'CONST'
        B  ENDS
        C  SEGMENT  'DATA'
        C  ENDS
        D  SEGMENT  STACK  'STACK'
        D  ENDS
        E  SEGMENT  'MEMORY'
        E  ENDS
```

You should be careful to declare all classes to be used in your program in this module. If you do not, you lose absolute control over the ordering of classes.

Also, if you want Memory combine type to be loaded as the last segments of your program, you can use this method. Simply add MEMORY between SEGMENT and 'MEMORY' in the E segment line above. Note, however, that these segments are loaded last only because you imposed this control on them, not because of any inherent capability in the linker or assembler operations.

Files That MS-LINK Uses

MS-LINK works with one or more input files, produces two output files, may create a virtual memory file, and may be directed to search one to eight library files. For each type of file, the user may give a three part file specification. The format for MS-LINK file specifications is:

drv:filename.ext

where: _drv:_ is the drive designation. Permissible drive designations for MS-LINK are A: through O:. The colon is always required as part of the drive designation.

filename is any legal filename of one to eight characters.

.ext is an one to three character extension to the filename. The period is always required as part of the extension.

Input Files

If no extensions are given in the input (Object) file specifications, MS-LINK recognizes by default:

File	Default Extension
Object	.OBJ
Library	.LIB

Output Files

MS-LINK appends to the output (Run and List) files the following default extensions:

File	Default Extension
Run	.EXE (may not be overridden)
List	.MAP (may be overridden)

VM.TMP File

MS-LINK uses available memory for the link session. If the files to be linked create an output file that exceeds available memory, MS-LINK creates a temporary file and names it VM.TMP. If MS-LINK needs to create VM.TMP, it displays the message:

```
        VM.TMP has been created.
        Do not change diskette in drive, <drv:>
```

Once this message is displayed, the user must not remove the diskette from the default drive until the link session ends. If the diskette is removed, the operation of MS-LINK is unpredictable, and MS-LINK might return the error message:

```
        Unexpected end of file on VM.TMP
```

MS-LINK uses VM.TMP as a virtual memory. The contents of VM.TMP are subsequently written to the file named following the Run File: prompt. VM.TMP is a working file only and is deleted at the end of the linking session.

WARNING

> Do not use VM.TMP as a file name for any file. If the user has a file named VM.TMP on the default drive and MS-LINK requires the VM.TMP file, MS-LINK will delete the VM.TMP on disk and create a new VM.TMP. Thus, the contents of the previous VM.TMP file will be lost.

CHAPTER 1

RUNNING MS-LINK

Running MS-LINK requires two types of commands: a command to invoke MS-LINK and answers to command prompts. In addition, six switches control alternate MS-LINK features. Usually, the user will enter all the commands to MS-LINK on the terminal keyboard. As an option, answers to the command prompts and any switches may be contained in a Response File. Some special command characters are provided to assist the user while entering linker commands.

1.1 INVOKING MS-LINK

MS-LINK may be invoked three ways. By the first method, the user enters the commands as answers to individual prompts. By the second method, the user enters all commands on the line used to invoke MS-LINK. By the third method, the user creates a Response File that contains all the necessary commands.

Summary of Methods to invoke MS-LINK

Method 1	LINK
Method 2	LINK <filenames>[/switches]
Method 3	LINK @<filespec>

1.1.1 Method 1: LINK

Enter:

```
        LINK
```

MS-LINK will be loaded into memory. Then, MS-LINK returns a series of four text prompts that appear one at a time. The user answers the prompts as commands to MS-LINK to perform specific tasks.

At the end of each line, you may enter one or more switches, each of which must be preceded by a slash mark. If a switch is not included, MS-LINK defaults to not performing the function described for the switches in the chart below.

The command prompts are summarized here and described in more detail in Section 2.2, COMMAND PROMPTS. Following the summary of prompts is a summary of switches, which are described in more detail in Section 2.3, Switches.

PROMPT	RESPONSES
Object Modules [.OBJ]:	List .OBJ files to be linked, separated by a blank spaces or plus signs (+). If plus sign is last character entered, prompt will reappear. (no default: response required)
Run File [Object-file.EXE]:	List filename for executable object code. (default: first-Object-filename.EXE)
List File [Run-file.MAP]:	List filename for listing (default: RUN filename)
Libraries [ ]:	List filenames to be searched, separated by blank spaces or plus signs (+). If plus sign is last character entered, prompt will reappear. (default: no search)

SWITCH	ACTION
/DSALLOCATE	Load data at high end of Data Segment. Required for Pascal and FORTRAN programs.
/HIGH	Place Run file as high as possible in memory. Do not use with Pascal or FORTRAN programs.
/LINENUMBERS	Include line numbers in List file.
/MAP	List all global symbols with definitions.
/PAUSE	Halt linker session and wait for carriage return key.
/STACK:<number>	Set fixed stack size in Run file.

Command Characters

MS-LINK provides three command characters.

+ Use the plus sign (+) to separate entries and to extend the current physical line following the Object Modules and Libraries prompts. (A blank space may be used to separate object modules.) To enter a large number of responses (each which may also be very long), enter an plus sign/carriage return at the end of the physical line (to extend the logical line). If the plus sign/carriage return is the last entry following these two prompts, MS-LINK will prompt the user for more modules names. When the Object Modules or Libraries prompt appears again, continue to enter responses. When all the modules to be linked have been listed, be sure the response line ends with a module name and a carriage return and not a plus sign/carriage return.

Example:

```
Object Modules [.OBJ]: FUN TEXT TABLE
CARE+<CR>
Object Modules [.OBJ]:
FOO+FLIPFLOP+JUNQUE+<CR>
Object Modules [.OBJ]: CORSAIR<CR>
```

; Use a single semicolon (;) followed immediately by a carriage return at any time after the first prompt (from Run File on) to select default responses to the remaining prompts. This feature saves time and overrides the need to enter a series of carriage returns.

NOTE

Once the semicolon has been entered, the user can no longer respond to any of the prompts for that link session. Therefore, do not use the semicolon to skip over some prompts. For this, use carriage return.

Example:

```
Object Modules [.OBJ]:  FUN TEXT TABLE CARE<CR>
Run Module [FUN.EXE]:  ;<CR>
```

The remaining prompts will not appear, and MS-LINK will use the default values (including FUN.MAP for the List File).

Control-C Use Control-C at any time to abort the link session. If you enter an erroneous response, such as the wrong filename or an incorrectly spelled filename, you must press Control-C to exit MS-LINK then reinvoke MS-LINK and start over. If the error has been typed but not entered, you may delete the erroneous characters, but for that line only.

1.1.2 Method 2: LINK <filenames>[/switches]

Enter:

```
LINK <object-list>,<runfile>,<listfile>,<lib-list>
                                          [/switch...]
```

The entries following LINK are responses to the command prompts. The entry fields for the different prompts must be separated by commas.

where: object list is a list of object modules, separated by plus signs

runfile is the name of the file to receive the executable output

listfile is the name of the file to receive the listing

lib list is a list of library modules to be searched

/switch are optional switches, which may be placed following any of the response entries (just before any of the commas or after the <lib list>, as shown).

To select the default for a field, simply enter a second comma without spaces in between (see the example below).

Example

```
LINK FUN+TEXT+TABLE+CARE/P/M,,FUNLIST,COBLIB.LIB
```

This example causes MS-LINK to be loaded, then causes the object modules FUN.OBJ, TEXT.OBJ, TABLE.OBJ, and CARE.OBJ to be loaded. MS-LINK then pauses (caused by the /P switch). When the user presses any key, MS-LINK links the object modules, produces a global symbol map (the /M switch), defaults to FUN.EXE run file, creates a list file named FUNLIST.MAP, and searches the library file COBLIB.LIB.

1.1.3 Method 3: LINK @<filespec>

Enter:

```
LINK @<filespec>
```

where: filespec is the name of a Response File. A Response File contains answers to the MS-LINK prompts (shown under method 1 for invoking), and may also contain any of the switches. Method 3 permits the user to conduct the MS-LINK session without interactive (direct) user responses to the MS-LINK prompts.

IMPORTANT

> Before using method 3 to invoke MS-LINK, the user must first create the Response File.

A Response File has text lines, one for each prompt. Responses must appear in the same order as the command prompts appear.

Use switches and Special Command Characters in the Response File the same way as they are used for responses entered on the terminal keyboard.

When the MS-LINK session begins, each prompt will be displayed in turn with the responses from the response file. If the response file does not contain answers for all the prompts, either in the form of filenames or the semicolon special character or carriage returns, MS-LINK will, after displaying the prompt which does not have a response, wait for the user to enter a legal response. When a legal response has been entered, MS-LINK continues the link session.

Example:

```
FUN TEXT TABLE CARE
/PAUSE/MAP
FUNLIST
COBLIB.LIB
```

This Response File will cause MS-LINK to load the four Object modules. MS-LINK will pause before creating and producing a public symbol map to permit the user to swap diskettes (see discussion under /PAUSE in Section 2.3, Switches, before using this feature). When the user presses any key, the output files will be named FUN.EXE and FUNLIST.MAP, MS-LINK will search the library file COBLIB.LIB, and will use the default settings for the flags.

1.2 COMMAND PROMPTS

MS-LINK is commanded by entering responses to four text prompts. When you have entered a response to the current prompt, the next appears. When the last prompt has been answered, MS-LINK begins linking automatically without further command. When the link session is finished, MS-LINK exits to the operating system. When the operating system prompt is displayed, MS-LINK has finished successfully. If the link session is unsuccessful, MS-LINK returns the appropriate error message.

MS-LINK prompts the user for the names of object, run, list files, and for libraries. The prompts are listed in their order of appearance. For prompts which can default to preset responses, the default response is shown in square brackets ([]) following the prompt. The Object Modules: prompt is followed by only a filename extension default response because it has no preset filename response and requires a filename from the user.

<u>Object</u> <u>Modules</u> <u>[.OBJ]:</u>

Enter a list of the object modules to be linked. MS-LINK assumes by default that the filename extension is .OBJ. If an object module has any other filename extension, the extension must be given here. Otherwise, the extension may be omitted.

Modules must be separated by plus signs (+).

Remember that MS-LINK loads Segments into Classes in the order encountered (see Section 1.2, DEFINITIONS). Use this information for setting the order in which the object modules are entered.

Run File [First-Object-filename.EXE]:

The filename entered will be created to store the Run (executable) file that results from the link session. All Run files receive the filename extension .EXE, even if the user specifies an extension (the user specified extension is ignored).

If no response is entered to the Run File: prompt, MS-LINK uses the first filename entered in response to the Object Modules: prompt as the RUN filename.

Example:

```
    Run File [FUN.EXE]:  B:PAYROLL/P
```

This response directs MS-LINK to create the Run file PAYROLL.EXE on drive B:. Also, MS-LINK will pause, which allows the user to insert a new diskette to receive the Run file.

List File [Run-Filename.MAP]:

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 default response is the Run filename with the default filename extension .MAP.

Libraries [ ]:

The valid responses are one to eight library filenames or simply a carriage return. (A carriage return only means no library search.) Library files must have been created by a library utility. MS-LINK assumes by default that the filename extension is .LIB for library files.

Library filenames must be separated by blank spaces or plus signs (+).

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

If MS-LINK cannot find a library file on the diskettes in the disk drives, it returns the message:

```
Cannot find library <library-name>
Enter new drive letter:
```

Simply press the letter for the drive designation (for example B).

MS-LINK does not search within each library file sequentially. MS-LINK uses a method called dictionary indexed library search. This means that MS-LINK finds definitions for external references by index access rather than searching from the beginning of the file to the end for each reference. This indexed search reduces substantially the link time for any sessions involving library searches.

1.3 SWITCHES

The six switches control alternate linker functions. Switches must be entered at the end of a prompt response, regardless of which method is used to invoke MS-LINK. Switches may be grouped at the end of any one of the responses, or may be scattered at the end of several. If more than one switch is entered at the end of one response, each switch must be preceded by the slash mark (/).

All switches may be abbreviated, from a single letter through the whole switch name. The only restriction is that an abbreviation must be a sequential sub-string from the first letter through the last entered; no gaps or transpositions are allowed. For example:

Legal	Illegal
/D	/DSL
/DS	/DAL
/DSA	/DLC
/DSALLOCA	/DSALLOCT

/DSALLOCATE

Use of the /DSALLOCATE switch directs MS-LINK to load all data (DGroup) at the high end of the Data Segment. Otherwise, MS-LINK loads all data at the low end of the Data Segment. At runtime, the DS pointer is set to the lowest possible address and allows the entire DS segment to be used. Use of the /DSALLOCATE switch in combination with the default load low (that is, the /HIGH switch is not used), permits the user application to allocate dynamically any available memory below the area specifically allocated within DGroup, yet to remain addressable by the same DS pointer. This dynamic allocation is needed for Pascal and FORTRAN programs.

NOTE

The user's application program may dynamically allocate up to 64K bytes (or the actual amount available) less the amount allocated within DGroup.

/HIGH

Use of the /HIGH switch causes MS-LINK to place the Run image as high as possible in memory. Otherwise, MS-LINK places the Run file as low as possible.

IMPORTANT

> Do not use the /HIGH switch with Pascal or FORTRAN programs.

/LINENUMBERS

Use of the /LINENUMBERS switch directs MS-LINK to include in the List file the line numbers and addresses of the source statements in the input modules. Otherwise, line numbers are not included in the List file.

NOTE

Not all compilers produce object modules that contain line number information. In these cases, of course, MS-LINK cannot include line numbers.

/MAP

/MAP directs MS-LINK to list all public (global) symbols defined in the input modules. If /MAP is not given, MS-LINK will list only errors (which includes undefined globals).

The symbols are listed alphabetically. For each symbol, MS-LINK lists its value and its segment:offset location in the Run file. The symbols are listed at the end of the List file.

/PAUSE

The /PAUSE switch causes MS-LINK to pause in the link session when the switch is encountered. Normally, MS-LINK performs the linking session without stop from beginning to end. This allows the user to swap the diskettes before MS-LINK outputs the Run (.EXE) file.

When MS-LINK encounters the /PAUSE switch, it displays the message:

```
About to generate .EXE file
Change disks <hit any key>
```

MS-LINK resumes processing when the user presses any key.

CAUTION

> Do not swap the diskette which will receive the List file, or the diskette used for the VM.TMP file, if created.

/STACK:<number>

number represents anv positive numeric value (in hexadecimal radix) up to 65536 bytes. If the /STACK switch is not used for a link session, MS-LINK calculates the necessary stack size automatically.

If a value from 1 to 511 is entered, MS-LINK uses 512.

All compilers and assemblers should provide information in the object modules that allow the linker to compute the required stack size.

At least one object (input) module must contain a stack allocation statement. If not, MS-LINK will return a WARNING: NO STACK STATEMENT error message.

CHAPTER 2

ERROR MESSAGES

All errors cause the link session to abort. Therefore, after the cause is found and corrected, MS-LINK must be rerun.

ATTEMPT TO ACCESS DATA OUTSIDE OF SEGMENT BOUNDS, POSSIBLY BAD OBJECT MODULE
Cause: probably a bad object file

BAD NUMERIC PARAMETER
Cause: numeric value not in digits

CANNOT OPEN TEMPORARY FILE
Cause: MS-LINK is unable to create the file VM.TMP because the disk directory is full.
Cure: insert a new diskette. Do not change the diskette that will receive the list.MAP file.

ERROR: DUP RECORD TOO COMPLEX
Cause: DUP record in assembly language module is too complex.
Cure: simplify DUP record in assembly language program.

ERROR: FIXUP OFFSET EXCEEDS FIELD WIDTH
Cause: an assembly language instruction refers to an address with a short instruction instead of a long instruction.
Cure: edit assembly language source and reassemble

INPUT FILE READ ERROR
Cause: probably a bad object file

INVALID OBJECT MODULE

Cause: object module(s) incorrectly formed or incomplete (as when assembly was stopped in the middle).

SYMBOL DEFINED MORE THAN ONCE

Cause: MS-LINK found two or more modules that define a single symbol name.

PROGRAM SIZE OR NUMBER OF SEGMENTS EXCEEDS CAPACITY OF LINKER

Cause: the total size may not exceed 384K bytes and the number of segments may not exceed 255

REQUESTED STACK SIZE EXCEEDS 64K

Cure: specify a size ≤ 64K bytes with the /STACK switch

SEGMENT SIZE EXCEEDS 64K

64K bytes is the addressing system limit.

SYMBOL TABLE CAPACITY EXCEEDED

Cause: very many, very long names entered; exceeding approximately 25K bytes.

TOO MANY EXTERNAL SYMBOLS IN ONE MODULE

The limit is 256 external symbols per module

TOO MANY GROUPS

The limit is 10 Groups

TOO MANY LIBRARIES SPECIFIED

The limit is 8.

TOO MANY PUBLIC SYMBOLS

The limit is 1024.

TOO MANY SEGMENTS OR CLASSES

The limit is 256 (Segments and Classes taken together)

UNRESOLVED EXTERNALS: <list>
The external symbols listed have no defining module among the modules or libraries files specified.

VM READ ERROR
Cause: a disk problem; not MS-LINK caused.

WARNING: NO STACK SEGMENT
Cause: none of the object modules specified contains a statement allocating stack space, but the user entered the /STACK switch.

WARNING: SEGMENT OF ABSOLUTE OR UNKNOWN TYPE
Cause: a bad object module or an attempt to link modules MS-LINK cannot handle (e.g., an absolute object module).

WRITE ERROR IN TMP FILE
Cause: no more disk space remaining to expand VM.TMP file

WRITE ERROR ON RUN FILE
Cause: usually, not enough disk space for Run file

INDEX

MS-LIB library manager

Microsoft, Inc.
Microsoft Building
10700 Northup Way
Bellevue, WA 98004

LIMITED WARRANTY

MICROSOFT, Inc. shall have no liability or responsibility to purchaser or to any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this product. This product will be exchanged within twelve months from date of purchase if defective in manufacture, labeling, or packaging, but except for such replacement the sale or subsequent use of this program is without warranty or liability.

THE ABOVE IS A LIMITED WARRANTY AND THE ONLY WARRANTY MADE BY MICROSOFT, INC. ANY AND ALL WARRANTIES FOR MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY EXCLUDED.

To report software bugs or errors in the documentation, please complete and return the Problem Report at the back of this manual.

8407C-100-00

System Requirements

The MS-LIB Library Manager requires:

38K bytes of memory minimum:
        28K bytes for code
        10K bytes for run space

1 disk drive
        1 disk drive if and only if output is sent to the same physical diskette from which the input was taken. None of the utility programs in this package allow time to swap diskettes during operation on a one-drive configuration. Therefore, two disk drives is a more practical configuration.

Contents

INTRODUCTION

Features and Benefits

MS-LIB creates and modifies library files that are used with Microsoft's MS-LINK Linker Utility. MS-LIB can add object files to a library, delete modules from a library, or extract modules from a library and place the extracted modules into separate object files.

MS-LIB provides a means of creating either general or special libraries for a variety of programs or for specific programs only. With MS-LIB you can create a library for a language compiler, or you can create a library for one program only, which would permit very fast linking and possibly more efficient execution.

You can modify individual modules within a library by extracting the modules, making changes, then adding the modules to the library again. You can also replace an existing module with a different module or with a new version of an existing module.

The command scanner in MS-LIB is the same as the one used in Microsoft's MS-LINK, MS-Pascal, MS-FORTRAN, and other 16-bit Microsoft products. If you have used any of these products, using MS-LIB is familiar to you. Command syntax is straightforward, and MS-LIB prompts you for any of the commands it needs that you have not supplied. There are no surprises in the user interface.

Overview of MS-LIB Operation

MS-LIB performs two basic actions: it deletes modules from a library file, and it changes object files into modules and appends them to a library file. These two actions underlie five library manager functions:

delete a module

extract a module and place it in a separate object file

append an object file as a module of a library

replace a module in the library file with a new module

create a library file

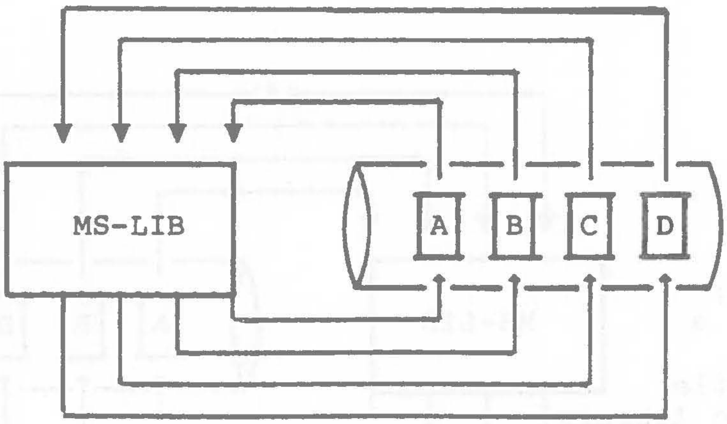
During each library session, MS-LIB first deletes or extracts modules, then appends new ones. In a single operation, MS-LIB reads each module into memory, checks it for consistency, and writes it back to the file. If you delete a module, MS-LIB reads in that module but does not write it back to the file. When MS-LIB writes back the next module to be retained, it places the module at the end of the last module written. This procedure effectively "closes up" the disk space to keep the library file from growing larger than necessary. When MS-LIB has read through the whole library file, it apends any new modules to the end of the file. Finally, MS-LIB creates the index, which MS-LINK uses to find modules and symbols in the library file, and outputs a cross reference listing of the PUBLIC symbols in the library, if you request such a listing. (Building the library index may take some extra time, up to 20 second in some cases.)

For example:

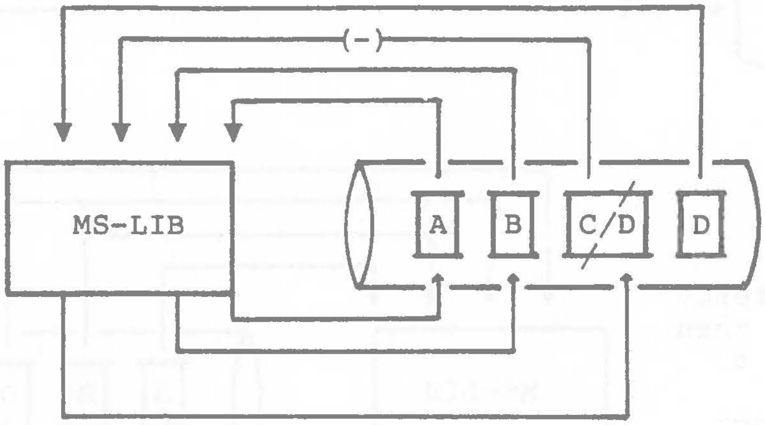
```
LIB PASCAL+HEAP-HEAP;
```

first deletes the library module HEAP from the library file, then adds the file HEAP.OBJ as the last module in the library. This order of execution prevents confusion in MS-LIB when a new version of a module replaces a version in the library file. Note that the replace function is simply the delete-append functions in succession. Also note that you can specify delete, append, or extract functions in any order; the order is insignificant to the MS-LIB command scanner.

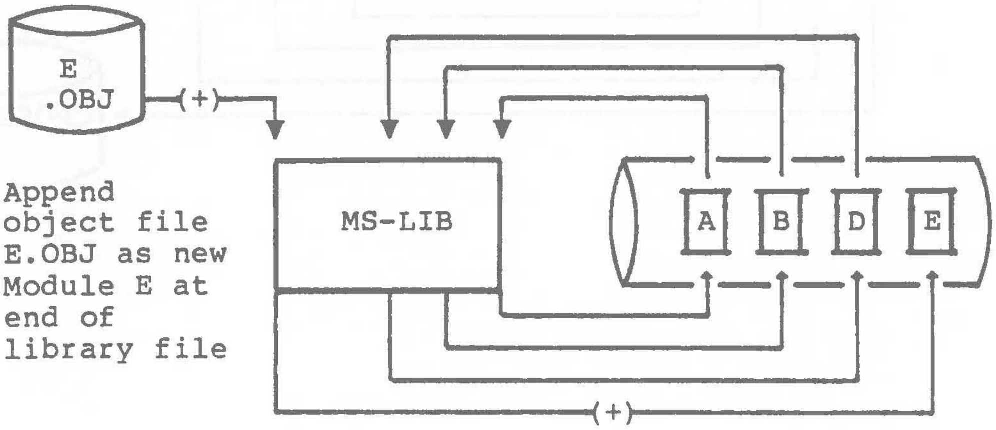
Consistency Check only



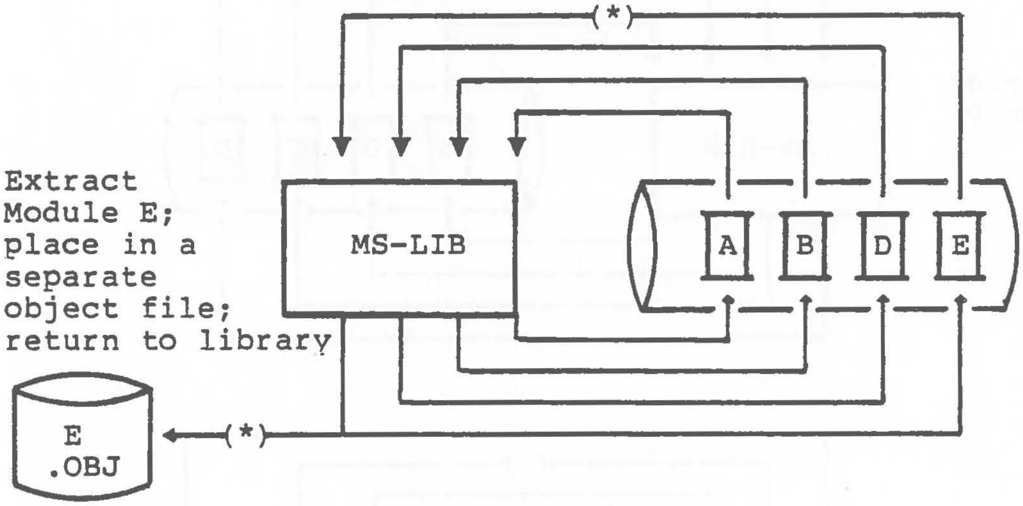
Delete Module C; Module D written to space of Module C



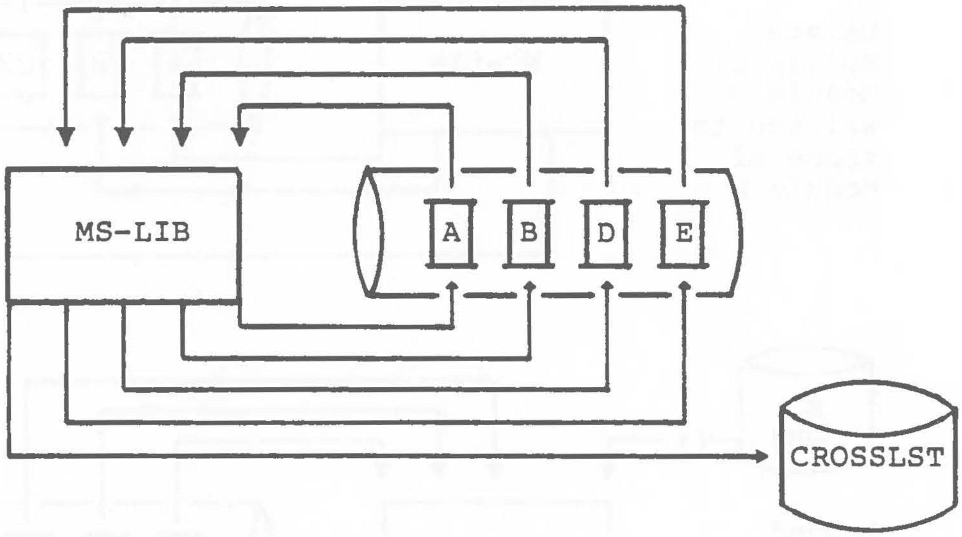
Append object file E.OBJ as new Module E at end of library file



Extract Module E; place in a separate object file; return to library



Consistency Check, then output a cross reference listing of PUBLIC symbols



CHAPTER 1

RUNNING MS-LIB

Running MS-LIB requires two types of commands: a command to invoke MS-LIB and answers to command prompts. Usually you will enter all the commands to MS-LIB on the terminal keyboard. As an option, answers to the command prompts may be contained in a Response File. Some special command characters exist. Some are used as a required part of MS-LIB commands. Others assist you while entering MS-LIB commands.

1.1 INVOKING MS-LIB

MS-LIB may be invoked three ways. By the first method, you enter the commands as answers to individual prompts. By the second method, you enter all commands on the line used to invoke MS-LIB. By the third method, you create a Response File that contains all the necessary commands.

Summary of Methods to invoke MS-LIB

Method 1	LIB
Method 2	LIB <library><operations>,<listing>
Method 3	LIB @<filespec>

1.1.1 Method 1: LIB

Enter:

```
        LIB
```

MS-LIB will be loaded into memory. Then, MS-LIB returns a series of three text prompts that appear one at a time. You answer the prompts as commands to MS-LIB to perform specific tasks.

The Command Prompts and Command Characters are summarized here. The Command Prompts and Command Characters are described fully in Sections 1.2 and 1.3.

Summary of Command Prompts

PROMPT	RESPONSES
Library file:	List filename of library to be manipulated (default: filename extension .LIB)
Operation:	List command character(s) followed by module name(s) or object filename(s) (default action: no changes. default object filename extension: .OBJ)
List file:	List filename for a cross reference listing file (default: NUL; no file)

Summary of Command Characters

Character	Action
+	Append an object file as the last module
-	Delete a module from the library
*	Extract a module and place in an object file
;	Use default responses to remaining prompts
&	Extend current physical line; repeat command prompt
Control-C	Abort library session.

1.1.2 Method 2: LIB <library><operations>,<listing>

Enter:

```
LIB <library><operations>,<listing>
```

The entries following LIB are responses to the command prompts. The _library_ and _operations_ fields and all operations entries must be separated by one of the command characters plus, minus, and asterisk (+, -, *). If a cross reference listing is wanted, the name of the file must be separated from the last operations entry by a comma.

where: _library_ is the name of a library file. MS-LIB assumes that the filename extension is .OBJ, which you may override by specifying a different extension. If the filename given for the _library_ field does not exist, MS-LIB will prompt you:

```
Library file does not exist.  Create?
```

Enter Yes (or any response beginning with Y) to create a new library file. Enter No (or any other response not beginning with Y) to abort the library session.

operations is deleting a module, appending an object file as a module, or extracting a module as an object file from the library file. Use the three command characters plus (+), minus (-), and asterisk (*) to direct MS-LIB what to do with each module or object file.

listing is the name of the file you want to receive the cross reference listing of PUBLIC symbols in the modules in the library. The list is compiled after all module manipulation has taken place.

To select the default for remaining field(s), you may enter the semicolon command character.

If you enter a Librarv filename followed immediately by a semicolon, MS-LIB will read through the library file and perform a consistency check. No changes will be made to the modules in the library file.

If you enter a Library filename followed immediately by a comma and a List filename, MS-LIB will perform its consistency check of the library file, then produce the cross reference listing file.

Example

```
LIB PASCAL-HEAP+HEAP;
```

This example causes MS-LIB to delete the module HEAP from the library file PASCAL.LIB, then append the object file HEAP.OBJ as the last module of PASCAL.LIB (the module will be named HEAP).

If you have many operations to perform during a library session, use the ampersand (&) command character to extend the line so that you can enter additional object filenames and module names. Be sure to always include one of the command characters for operations (+, -, *) before the name of each module or object filename.

Example

```
    LIB PASCAL<CR>
```

causes MS-LIB to perform a consistency check of the library file PASCAL.LIB. No other action is performed.

Example

```
    LIB PASCAL,PASCROSS.PUB
```

causes MS-LIB to perform a consistency check of the library file PASCAL.LIB, then cutput a cross reference listing file named PASCROSS.PUB.

1.1.3 Method 3: LIB @<filespec>

Enter:

```
LIB @<filespec>
```

where: filespec is the name of a Response File. A Response File contains answers to the MS-LIB prompts (summarized under method 1 for invoking and described fully in Section 1.2). Method 3 permits you to conduct the MS-LIB session without interactive (direct) user responses to the MS-LIB prompts.

IMPORTANT

> Before using method 3 to invoke MS-LIB, you must first create the Response File.

A Response File has text lines, one for each prompt. Responses must appear in the same order as the command prompts appear.

Use Command Characters in the Response File the same way as they are used for responses entered on the terminal keyboard.

When the library session begins, each prompt will be displayed in turn with the responses from the response file. If the response file does not contain answers for all the prompts, MS-LIB will use the default responses (no changes to the modules currently in the library file for Operation, and no cross reference listing file created).

If you enter a Library filename followed immediately by a semicolon, MS-LIB will read through the library file and perform a consistency check. No changes will be made to the modules in the library file.

If you enter a Library filename then only a carriage return of Operations then a comma and a List filename, MS-LIB will perform its consistency check of the library file, then produce the cross reference listing file.

Example:

```
PASCAL<CR>
+CURSOR+HEAP-HEAP*FOIBLES<CR>
CROSSLST<CR>
```

This Response File will cause MS-LIB to delete the module HEAP from the PASCAL.LIB library file, extract the module FOIBLES and place in an object file named FOIBLES.OBJ, then append the object files CURSOR.OBJ and HEAP.OBJ as the last two modules in the library. Then, MS-LIB will create a cross reference file named CROSSLST.

1.2 COMMAND PROMPTS

MS-LIB is commanded by entering responses to three text prompts. When you have entered your response to the current prompt, the next appears. When the last prompt has been answered, MS-LIB performs its library management functions without further command. When the library session is finished, MS-LIB exits to the operating system. When the operating system prompt is displayed, MS-LIB has finished the library session successfully. If the library session is unsuccessful, MS-LIB returns the appropriate error message.

MS-LIB prompts you for the name of the library file, the operation(s) you want to perform, and the name you want to give to a cross reference listing file, if any.

<u>Library file:</u>

Enter the name of the library file that you want to manipulate. MS-LIB assumes that the filename extension is .LIB. You can override this assumption by giving a filename extension when you enter the library filename. Because MS-LIB can manage only one library file at a time, only one filename is allowed in response to this prompt. Additional responses, except the semicolon command character, are ignored.

If you enter a library filename and follow it immediately with a semicolon command character, MS-LIB will perform a consistency check only, then return to the operating system. Any errors in the file will be reported.

If the filename you enter does not exist, MS-LIB returns the prompt:

```
Library file does not exist.  Create?
```

You must enter either Yes or No, in either upper or lower (or mixed) case. Actually, MS-LIB checks the response for the letter Y as the first charcter. If any other character is entered first, MS-LIB terminates and returns to the operating system.

Operation:

Enter one of the three command characters for manipulating modules (+, -, *), followed immediately (no space) by the module name or the object filename. Plus sign appends an object file as the last module in the library file (see further discussion under the description of plus sign below). Minus sign deletes a module from the library file. Asterisk extracts a module from the library and places it in a separate object file with the filename taken from the module name and a filename extension .OBJ.

When you have a large number of modules to manipulate (more than can be typed on one line), enter an ampersand (&) as the last character on the line. MS-LIB will repeat the Operation prompt, which permits you to enter additional module names and object filenames.

MS-LIB allows you to enter operations on modules and object files in any order you want.

More information about order of execution and what MS-LIB does with each module is given in the descriptions of each Command Character.

List file:

If you want a cross reference list of the PUBLIC symbols in the modules in the library file after your manipulations, enter a filename in which you want MS-LIB to place the cross reference listing. If you do not enter a filename, no cross reference listing is generated (a NUL file).

The response to the List file prompt is a file specification. Therefore, you can specify, along with the filename, a drive (or device) designation and a filename extension. The List file is not given a default filename extension. If you want the file to have a filename extension, you must specify it when entering the filename.

The cross reference listing file contains two lists. The first list is an alphabetical listing of all PUBLIC symbols. Each symbol name is followed by the name of its module. The second list is an alphabetical list of the modules in the library. Under each module name is an alphabetical listing of the PUBLIC symbols in that module.

1.3 COMMAND CHARACTERS

MS-LIB provides six command characters: three of the command characters are required in responses to the Operation prompt; the other three command characters provide you additional helpful commands to MS-LIB.

+ The plus sign followed by an object filename appends the object file as the last module in the library named in reponse to the Library file prompt. When MS-LIB sees the plus sign, it assumes that the filename extension is .OBJ. You may override this assumption by specifying a different filename extension.

MS-LIB strips the drive designation and the extension from the object file specification, leaving only the filename. For example, if the object file to be appended as a module to a library is:

```
B:CURSOR.OBJ
```

a response to the Operation prompt of:

```
+B:CURSOR.OBJ
```

causes MS-LIB to strip off the B: and the .OBJ, leaving only CURSOR, which becomes a module named CURSOR in the library.

NOTE

The distinction between an object file and a module (or object module) is that the file possesses a drive designation (even if it is default drive) and a filename extension. Object modules possess neither of these.

- The minus sign followed by a module name deletes that module from the library file. MS-LIB then "closes up" the file space left empty by the deletion. This cleanup action keeps the library file from growing larger than necessary with empty space. Remember that new modules, even replacement modules are added to the end of the file, not stuffed into space vacated by deleting modules.

* The asterisk followed by a module name extracts that module from the library file and places it into a separate object file. The module will still exist in the library (extract means, essentially, copy the module to a separate object file). The module name is used as the filename. MS-LIB adds the default drive designation and the filename extension .OBJ. For example, if the module to be extracted is:

```
CURSOR
```

and the current default disk drive is A:, a reponse to the Operation prompt of:

```
*CURSOR
```

causes MS-LIB to extract the module named CURSOR from the library file and to set it up as an object file with the file specification of:

```
default drive:CURSOR.OBJ
```

(The drive designation and filename extension cannot be overridden. You can, however, rename the file, giving a new filename extension, and/or copy the file to a new disk drive, giving a new filename and/or filename extension.)

; Use a single semicolon (;) followed immediately by a carriage return at any time after responding to the first prompt (from Library file on) to select default responses to the remaining prompts. This feature saves time and overrides the need to answer additional prompts.

NOTE

Once the semicolon has been entered, you can no longer respond to any of the prompts for that library session. Therefore, do not use the semicolon to skip over some prompts. For this, use carriage return.

Example:

```
Library file:  FUN <CR>
Operation:  +CURSOR;<CR>
```

The remaining prompt will not appear, and MS-LIB will use the default value (no cross reference file).

& Use the ampersand to extend the current physical line. This command character will only be needed for the Operation prompt. MS-LIB can perform many functions during a single library session. The number of modules you can append it limited only be disk space. The number of module you can replace or extract is also limited only by disk space. The number of modules you can delete is limited only by the number of modules in the library file. However, the line length for a response to any prompt is limited to the line length of your system. For a large number of responses to the Operation prompt, place an ampersand at the end of a line. MS-LIB will display the Operation prompt again, then enter more responses. You may use the ampersand character as many times as you need. For example:

```
Library file:  FUN<CR>
Operation:   +CURSOR-HEAP+HEAP*FOIBLES&
Operation:   *INIT+ASSUME+RIDE;<CR>
```

MS-LIB will delete the module HEAP, extract the modules FOIBLES and INIT (creating two files, FOIBLES.OBJ and INIT.OBJ), then append the object files CURSOR, HEAP, ASSUME, and RIDE. Note, however, that MS-LIB allows you to enter your Operation reponses in any order.

Control-C Use Control-C at any time to abort the library session. If you enter an erroneous response, such as the wrong filename or module name, or an incorrectly spelled filename or module name, you must press CTRL-C to exit MS-LIB then reinvoke MS-LIB and start over. If the error has been typed but not entered, you may delete the erroneous characters, but for that line only.

CHAPTER 2

ERROR MESSAGES

<symbol> is a multiply defined PUBLIC. Proceed?
Cause: two modules define the same public symbol. The user is asked to confirm the removal of the definition of the old symbol. A No response leaves the library in an undetermined state.
Cure: Remove the PUBLIC declaration from one of the object modules and recompile or reassemble.

Allocate error on VM.TMP
Cause: out of space

Cannot create extract file
Cause: no room in directory for extract file

Cannot create list file
Cause: No room in directory for library file

Cannot nest response file
Cause: '@filespec' in response (or indirect) file

Cannot open VM.TMP
Cause: no room for VM.TMP in disk directory

Cannot write library file
Cause: Out of space

Close error on extract file
Cause: out of space

Error: An internal error has occurred.
Contact Microsoft, Inc.

Fatal Error: Cannot open input file
Cause: Mistyped object file name

Fatal Error: Module is not in the library
Cause: trying to delete a module that is not in the library

Input file read error
Cause: bad object module or faulty disk

Invalid object module/library
Cause: bad object and/or library

Library Disk is full
Cause: no more room on diskette

Listing file write error
Cause: out of space

No library file specified
Cause: no response to Library File prompt

Read error on VM.TMP
Cause: disk not ready for read

Symbol table capacity exceeded
Cause: too many public symbols (about 30K chars in symbols)

Too many object modules
Cause: more than 500 object modules

Too many public symbols
Cause: 1024 public symbols maximum

Write error on library/extract file
Cause: Out of space

Write error on VM.TMP
Cause: out of space

INDEX

microsoft
MS-CREF
cross reference facility
manual

Microsoft, Inc.
Microsoft Building
10700 Northup Way
Bellevue, WA 98004

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft, Inc. The software described in this document is furnished under a license agreement or non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the MS-CREF Cross Reference Facility on cassette tape, disk, or any other medium for any purpose other than the purchaser's personal use.

LIMITED WARRANTY

MICROSOFT, Inc. shall have no liability or responsibility to purchaser or to any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this product. This product will be exchanged within twelve months from date of purchase if defective in manufacture, labeling, or packaging, but except for such replacement the sale or subsequent use of this program is without warranty or liability.

THE ABOVE IS A LIMITED WARRANTY AND THE ONLY WARRANTY MADE BY MICROSOFT, INC. ANY AND ALL WARRANTIES FOR MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY EXCLUDED.

To report software bugs or errors in the documentation, please complete and return the Problem Report at the back of this manual.

MS-CREF, MACRO-86, MS-LINK, MS-LIB, and MS-DOS (and its constituent program names EDLIN and DEBUG) are trademarks of Microsoft, Inc.

System Requirements

The MS-CREF Cross Reference Facility requires:

24K bytes of memory minimum:
- 14K bytes for code
- 10K bytes for run space

1 disk drive

1 disk drive if and only if output is sent to the same physical diskette from which the input was taken. None of the utility programs in this package allow time to swap diskettes during operation on a one-drive configuration. Therefore, two disk drives is a more practical configuration.

Contents

INTRODUCTION

Features and Benefits

The MS-CREF Cross Reference Facility can aid you in debugging your assembly language programs. MS-CREF produces an alphabetical listing of all the symbols in a special file produced by your assembler. With this listing, you can quickly locate all occurrences of any symbol in your source program by line number.

The MS-CREF produced listing is meant to be used with the symbol table produced by your assembler.

The symbol table listing shows the value of each symbol, and its type and length, and its value. This information is needed to correct erroneous symbol definitions or uses.

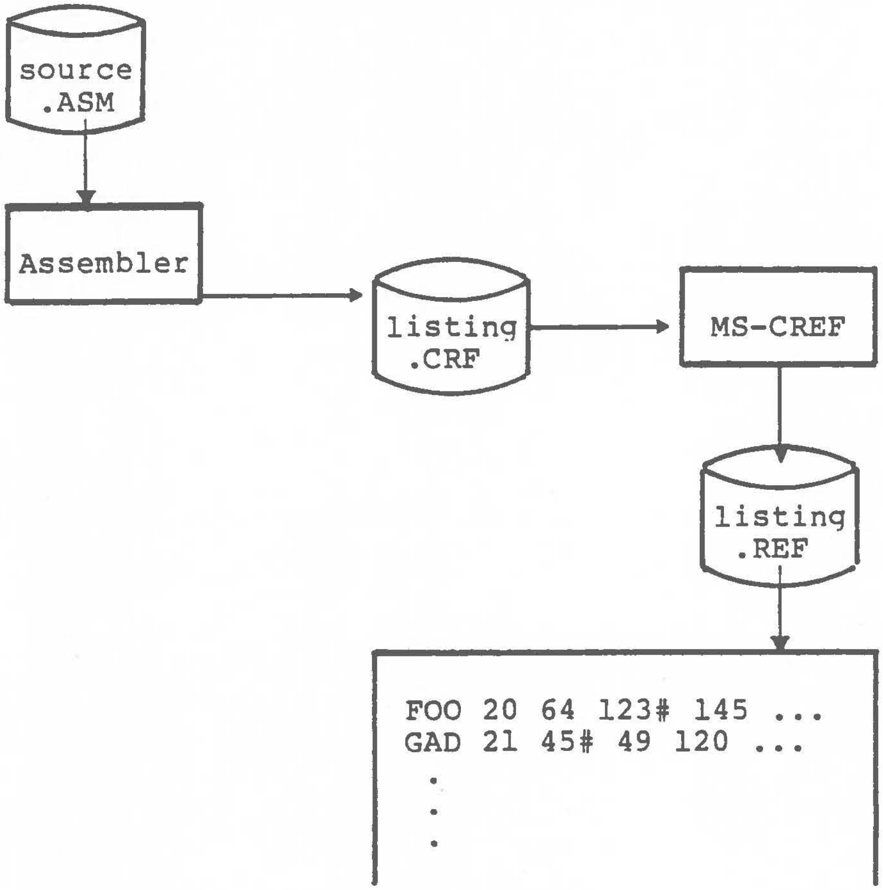
The cross reference listing produced by MS-CREF provides you the locations, speeding your search and allowing faster debugging.

Overview of MS-CREF Operation

MS-CREF produces a file with cross references for symbolic names in your program.

First, you must create a cross reference file with the assembler. Then, MS-CREF takes this cross reference file, which has the filename extension .CRF, and turns it into an alphabetical listing of the symbols in the file. The cross reference listing file is given the default filename extension .REF.

Beside each symbol in the listing, MS-CREF lists the line numbers in the source program where the symbol occurs in ascending sequence. The line number where the symbol is defined is indicated by a pound sign (#).



```
FOO 20 64 123# 145 ...
GAD 21 45# 49 120 ...
  .
  .
  .
```

CHAPTER 1

RUNNING MS-CREF

Running MS-CREF requires two types of commands: a command to invoke MS-CREF and answers to command prompts. You will enter all the commands to MS-CREF on the terminal keyboard. Some special command characters exist to assist you while entering MS-CREF commands.

Before you can use MS-CREF to create the cross reference listing, you must first have created a cross reference file using your assembler. This step is reviewed in Section 1.1.

1.1 CREATING A CROSS REFERENCE FILE

A cross reference file is created during an assembly session.

To create a cross reference file, answer the fourth assembler command prompt with the name of the file you want to receive the cross reference file.

The fourth assembler prompt is:

```
        Cross reference [NUL.CRF]:
```

If you do not enter a filename in response to this prompt, or if you in any other way use the default response to this prompt, the assembler will not create a cross reference file. Therefore, you must enter a filename. You may also specify which drive or device you want to receive the file and what filename extension you want the file to have, if different from .CRF. If you change the filename extension from .CRF to anything else, you must remember to specify the filename extension when naming the file in response to the first MS-CREF prompt (see Section 1.2.1).

When you have given a filename in response to the fourth assembler prompt, the cross reference file will be generated during the assembly session.

You are now ready to convert the cross reference file produced by the assembler into a cross reference listing using MS-CREF.

1.2 INVOKING MS-CREF

MS-CREF may be invoked two ways. By the first method, you enter the commands as answers to individual prompts. By the second method, you enter all commands on the line used to invoke MS-CREF.

Summary of Methods to invoke MS-CREF

Method 1	CREF
Method 2	CREF <crffile>,<listing>

1.2.1 Method 1: CREF

Enter:

```
        CREF
```

MS-CREF will be loaded into memory. Then, MS-CREF returns a series of two text prompts that appear one at a time. You answer the prompts to command MS-CREF to convert a cross reference file into a cross reference listing.

Command Prompts

Cross reference [.CRF]:

Enter the name of the cross reference file you want MS-CREF to convert into a cross reference listing. The name of the file is the name you gave your assembler when you directed it to produce the cross reference file.

MS-CREF assumes that the filename extension is .CRF. If you do not specify a filename extension when you enter the cross reference filename, MS-CREF will look for a file with the name you specify and the filename extension .CRF. If your cross reference file has a different extension, specify the extension when entering the filename.

See Chapter 3, Format of MS-CREF Compatible Files, for a description of what MS-CREF expects to see in the cross reference file. You will need this information only if your cross reference file was not produced by a Microsoft assembler.

Listing [crffile.REF]:

Enter the name you want the cross reference listing file to have. MS-CREF will automatically give the cross reference listing the filename extension .REF.

If you want you cross reference listing to have the same filename as the cross reference file but with the filename extension .REF, simply press the carriage return key when the Listing prompt appears. If you want your cross reference listing file to be named anything else and/or to have any other filename extension, you must enter a response following the Listing prompt.

If you want the listing file placed on a drive or device other than the default drive, specify the drive or device when entering your response to the Listing prompt.

Special Command Characters

; Use a single semicolon (;) followed immediately by a carriage return at any time after responding to the Cross reference prompt to select the default response to the Listing prompt. This feature saves time and overrides the need to answer the Listing prompt.

If you use the semicolon, MS-CREF gives the listing file the filename of the cross reference file and the default filename extension .REF.

Example:

```
Cross reference [.CRF]: FUN;
```

MS-CREF will process the cross reference file named FUN.CRF and output a listing file named FUN.REF.

Control-C Use Control-C at any time to abort the MS-CREF session. If you enter an erroneous response, (the wrong filename), or an incorrectly spelled filename, you must press Control-C to exit MS-CREF then reinvoke MS-CREF and start over. If the error has been typed but not entered, you may delete the erroneous characters, but for that line only.

1.2.2 Method 2: CREF <crffile>,<listing>

Enter:

```
        CREF <crffile>,<listing>
```

MS-CREF will be loaded into memory. Then, MS-CREF immediately procedes to convert your cross reference file into a cross reference listing.

The entries following CREF are responses to the command prompts. The crffile and listing fields must be separate by a comma.

where: crffile is the name of a cross reference file produced by your assembler. MS-CREF assumes that the filename extension is .CRF, which you may override by specifying a different extension. If the file named for the crffile does not exist, MS-CREF will display the message:

```
    Fatal I/O Error 110
    in File:  <crffile>.CRF
```

Control then returns to your operating system.

listing is the name of the file you want to receive the cross reference listing of symbols in your program.

To select the default filename and extension for the listing file, enter a semicolon after you enter the crffile name.

Example:

```
    CREF FUN;<CR>
```

This example causes MS-CREF to process the cross reference file FUN.CRF and to produce a listing file named FUN.REF.

To give the listing file a different name, extension, or destination, simply specify these differences when entering the command line.

```
    CREF FUN,B:WORK.ARG
```

This example causes MS-CREF to process the cross reference file named RUN.CRF and to produce a listing file named WORK.ARG, which will be placed on the diskette in drive B:.

1.3 FORMAT OF CROSS REFERENCE LISTINGS

The cross reference listing is an alphabetical list of all the symbols in your program.

Each page is headed with the title of the program or program module.

Then comes the list of symbols. Following each symbol name is a list of the line numbers where the symbol occurs in your program. The line number for the definition has a pound sign (#) appended to it.

On the next page is a cross reference listing as an example:

```
MS-CREF     (vers no.)     (date)

ENTX    PASCAL entry for initializing programs  ←comes from
                                                 TITLE directive

 Symbol Cross Reference      (# is definition)        Cref-1

AAAXQQ . . . . . . .      37#   38

BEGHQQ . . . . . . .      83    84#  154   176
BEGOQQ . . . . . . .      33   162
BEGXQQ . . . . . . .     113   126#  164   223

CESXQQ . . . . . . .      97    99#  129
CLNEQQ . . . . . . .      67    68#
CODE . . . . . . . .      37   182
CONST. . . . . . . .     104   104   105   110
CRCXQQ . . . . . . .      93    94#  210   215
CRDXQQ . . . . . . .      95    96#  216
CSXEQQ . . . . . . .      65    66#  149
CURHQQ . . . . . . .      85    86#  155

DATA . . . . . . . .      64#   64   100   110
DGROUP . . . . . . .     110#  111   111   111   127   153   171   172
DOSOFF . . . . . . .      98#  198   199
DOSXQQ . . . . . . .     184   204#  219

ENDHQQ . . . . . . .      87    88#  158
ENDOQQ . . . . . . .      33#  195
ENDUQQ . . . . . . .      31#  197
ENDXQQ . . . . . . .     184   194#
ENDYQQ . . . . . . .      32#  196
ENTGQQ . . . . . . .      30#  187
ENTXCM . . . . . . .     182#  183   221

FREXQQ . . . . . . .     169   170#  178

HDRFQQ . . . . . . .      71    72#  151
HDRVQQ . . . . . . .      73    74#  152
HEAP . . . . . . . .      42    44   110
HEAPBEG. . . . . . .      54#  153   172
HEAPLOW. . . . . . .      43   171

INIUQQ . . . . . . .      31   161

MAIN_STARTUP . . . .     109#  111   180
MEMORY . . . . . . .      42    48#   48    49   109   110

PNUXQQ . . . . . . .      69    70   150

RECEQQ . . . . . . .      81    82#
REFEQQ . . . . . . .      77    78#
REPEQQ . . . . . . .      79    80#
RESEQQ . . . . . . .      75    76#  148
```

```
SKTOP. . . . . . . . .        59#
SMLSTK . . . . . . . .       135    137#
STACK. . . . . . . . .        53#    53    60    110
STARTMAIN. . . . . . .       163    186#  200
STKBQQ . . . . . . . .        89     90#  146
STKHQQ . . . . . . . .        91     92#  160
```

CHAPTER 2

ERROR MESSAGES

All errors cause MS-CREF to abort. Control is returned to your operating system.

All error messages are displayed in the format:

```
        Fatal I/O Error <error number>
        in File:  <filename>
```

where: filename is the name of the file where the error occurs

error number is one of the numbers in the following list of errors.

Number	Error
101	Hard data error Unrecoverable disk I/O error
101	Device name error Illegal device specification (for example, X:FOO.CRF)
103	Internal error Report to Microsoft, Inc.
104	Internal error Report to Microsoft, Inc.
105	Device offline disk drive door open, no printer attached, and so on.
106	Internal error Report to Microsoft, Inc.
108	Disk full
110	File not found
111	Disk is write protected
112	Internal error Report to Microsoft, Inc.
113	Internal error Report to Microsoft, Inc.
114	Internal error Report to Microsoft, Inc.
115	Internal error Report to Microsoft, Inc.

CHAPTER 3

FORMAT OF MS-CREF COMPATIBLE FILES

MS-CREF will process files other than those generated by Microsoft's assembler as long as the file conforms to the format that MS-CREF expects.

3.1 GENERAL DESCRIPTION OF MS-CREF FILE PROCESSING

In essence, MS-CREF reads a stream of bytes from the cross reference file (or source file), sorts them, then emits them as a printable listing file (the .REF file). The symbols are held in memory as a sorted tree. References to the symbols are held in a linked list.

MS-CREF keeps track of line numbers in the source file by the number of end-of-line characters it encounters. Therefore, every line in the source file must contain at least an end-of-line character (see chart below).

MS-CREF attempts to place a heading at the top of every page of the listing. The name it uses as a title is the text passed by your assembler from a TITLE (or similar) directive in your source program. The title must be followed by a title symbol (see chart below). If MS-CREF encounters more than one title symbol in the source file, it uses the last title read for all page headings. If MS-CREF does not encounter a title symbol in the file, the title line on the listing is left blank.

3.2 FORMAT OF SOURCE FILES

MS-CREF uses the first three bytes of the source file as format specification data. The rest of the file is processed as a series of records that either begin or end with a byte that identifies the type of record.

First Three Bytes

(The PAGE directive in your assembler, which takes arguments for page length and line length, will pass this information to the cross reference file.)

First Byte
The number of lines to be printed per page (page length range is from 1 to 255 lines).

Second Byte
The number of characters per line (line length range is from 1 to 132 characters).

Third Byte
The Page Symbol (07) that tells MS-CREF that the two preceding bytes define listing page size.

If MS-CREF does not see these first three bytes in the file, it uses default values for page size (page length: 58 lines; line length: 80 characters).

Control Symbols

The two charts show the types of records that MS-CREF recognizes and the byte values and placement it uses to recognize record types.

Record have a Control Symbol (which identifies the record type) either as the first byte of the record or as the last byte.

Records That Begin with a Control Symbol

Byte value	Control Symbol	Subsequent Bytes
01	Reference symbol	Record is a reference to a symbol name (1 to 80 characters)
02	Define symbol	Record is a definition of a symbol name (1 to 80 characters)
04	End of line	(none)
05	End of file	1AH

Records That End with a Control Symbol

Byte value	Control Symbol	Preceding Bytes
06	Title defined	Record is title text (1 to 80 characters)
07	Page length/ line length	One byte for page length followed by one byte for line length

For all record types, the byte value represents a control character, as follows:

01	Control-A
02	Control-B
04	Control-D
05	Control-E
06	Control-F
07	Control-G

The Control Symbols are defined as follows:

Reference symbol
: Record contains the name of a symbol that is referenced. The name may be from 1 to 80 ASCII characters long. Additional characters are truncated.

Define symbol
: Record contains the name of a symbol that is defined. The name may be from 1 to 80 ASCII characters long. Additional characters are truncated.

End of line
: Record is an end of line symbol character only (04H or Control-D)

End of file
: Record is the end of file character (1AH)

Title defined
: ASCII characters of the title to be printed at the top of each listing page. The title may be from 1 to 80 characters long. Additional characters are truncated. The last title definition record encountered is used for the title placed at the top of all pages of the listing. If a title definition record is not encountered, the title line on the listing is left blank.

Page length/line length
: The first byte of the record contains the number of lines to be printed per page (range is from 1 to 255 lines). The second byte contains the number of characters to be printed per page (range is from 1 to 132 characters). The default page length is 58 lines. The default line length is 80 characters.

Summary of CRF File Record Contents

byte contents	length of record
|01|symbol name|	2-81 bytes
|02|symbol name|	2-81 bytes
|04|	1 byte
|05|1A|	2 bytes
|title text|06|	2-81 bytes
|PL|LL|07|	3 bytes

INDEX

m n	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	(null)	◀	(space)	0	@	P	`	p		┌		É			√	┏
1	╨	±	!	1	A	Q	a	q	☺	└	§	Å			π	┗
2	⊥	≥	"	2	B	R	b	r	☻	┘	Ä	å			×	┛
3	↕	≤	#	3	C	S	c	s	♥	┐	Ö	Æ			÷	┓
4	↨	≈	$	4	D	T	d	t	♦	│	Ü	Φ			Σ	┃
5	■	▽	%	5	E	U	e	u	♣	_	ä	æ			↑	━
6	□	½	&	6	F	V	f	v	♠	┼	ö	Ĳ			↓	╋
7	♪	^{2}	'	7	G	W	g	w	●	└	ü	ĳ			→	┣
8	≡	^{3}	(	8	H	X	h	x	◘	┘	ß	ò			←	┫
9	▶	^{4}	)	9	I	Y	i	y	○	┬	à	ì			μ	┳
A	↵	$_1$	*	:	J	Z	j	z	◙	┴	°				σ	┻
B	▼	$_2$	+	;	K	[	k	{	♂	↖	ç				φ	┠
C	▲	$_3$	,	<	L	\	l	¦	♀	↵	é				α	─
D	⇤	—	-	=	M	]	m	}	¿	⇥	Ù				β	┨
E	◆	▨	.	>	N	^	n	~	♟	⇤	è				γ	═
F	∟	▩	/	?	O	_	o	█	▌	▎	ē				£	█