Disk Operating System Version 4.00 Technical Reference

Programming Family
Application Programming

International Business Machines Corporation *Armonk, New York 10504*

IBM Program License Agreement

BEFORE OPENING THIS PACKAGE, YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS. OPENING THIS PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED AND YOUR MONEY WILL BE REFUNDED.

This is a license agreement and not an agreement for sale. IBM owns, or has licensed from the owner, copyrights in the Program. You obtain no rights other than the license granted you by this Agreement. Title to the enclosed copy of the Program, and any copy made from it, is retained by IBM. IBM licenses your use of the Program in the United States and Puerto Rico. You assume all responsibility for the selection of the Program to achieve your intended results and for the installation of, use of, and results obtained from, the Program.

The Section in the enclosed documentation entitled "License Information" contains additional information concerning the Program and any related Program Services.

LICENSE

You may:

1) use the Program on only one machine at any one time, unless permission to use it on more than one machine at any one time is granted in the License Information (Authorized Use);
2) make a copy of the Program for backup or modification purposes only in support of your Authorized Use. However, Programs marked "Copy Protected" limit copying;
3) modify the Program and/or merge it into another program only in support of your Authorized Use; and
4) transfer possession of copies of the Program to another party by transferring this copy of the IBM Program License Agreement, the License Information, and all other documentation along with at least one complete, unaltered copy of the Program. You must, at the same time, either transfer to such other party or destroy all your other copies of the Program, including modified copies or portions of the Program merged into other programs. Such transfer of possession terminates your license from IBM. Such other party shall be licensed, under the terms of this Agreement, upon acceptance of this Agreement by its initial use of the Program.

You shall reproduce and include the copyright notice(s) on all such copies of the Program, in whole or in part.

You shall not:

1) use, copy, modify, merge, or transfer copies of the Program except as provided in this Agreement;
2) reverse assemble or reverse compile the Program;
   and/or
3) sublicense, rent, lease, or assign the Program or any copy thereof.

LIMITED WARRANTY

Warranty details and limitations are described in the Statement of Limited Warranty which is available upon request from IBM, its Authorized Dealer or its approved supplier and is also contained in the License Information. IBM provides a three-month limited warranty on the media for all Programs. For selected Programs, as indicated on the outside of the package, a limited warranty on the Program is available. The applicable Warranty Period is measured from the date of delivery to the original user as evidenced by a receipt.

Certain Programs, as indicated on the outside of the package, are not warranted and are provided "AS IS."

Continued on inside back cover.

Disk Operating System Version 4.00 Technical Reference

Programming Family
Application Programming

First Edition (July 1988)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

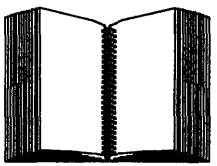
Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

© Copyright International Business Machines Corporation 1981, 1988
All rights reserved.

DOS 4.00 Library

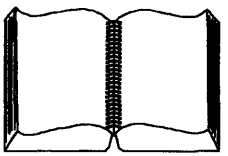
Getting Started With DOS 4.00

The first part of this book provides information you need to install DOS 4.00 and supplements the online information in the SELECT program. The second part introduces you to the DOS Shell.



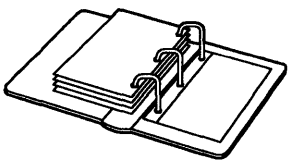
Using DOS 4.00

By using examples, this book explains how to manage your information from the command prompt, how to change the configuration of your system, and how to create and change batch files.



DOS 4.00 Command Reference

This book, an additional purchase item, provides detailed information on the commands used in DOS 4.00 and contains tables relating tasks to these commands.



DOS 4.00 Technical Reference and Application Programming

This book, an additional purchase item, is written for programmers who develop applications for IBM Personal Computers and Personal System/2®.

Personal System/2 is a registered trademark of the International Business Machines Corporation.

Preface

Audience

This book is written for programmers who develop applications for IBM Personal Computers and Personal System/2® computers.[1]

The program developer should be competent on the IBM Personal Computer and/or the Personal System/2 and should be familiar with DOS and at least one personal computer programming language.

Content

Some parts of the book have been rewritten and reorganized to make reference and system architecture information more concise and to present programming information in a task-oriented manner. Information about new and enhanced functions has been added, providing some guide material.

Related Publications

Other manuals that provide detailed information about DOS 4.00 are:

- *Getting Started with DOS 4.00*
- *Using DOS 4.00*
- *DOS 4.00 Command Reference*
- *IBM Keyboard Layouts for Your PC and PS/2 systems.*

---

[1] Personal System/2 is a registered trademark of the International Business Machines Corporation.

Contents

Part 1. Writing Programs

Part 2. Using the Programming Utilities

Part 3. Appendixes

Chapter 1. Introduction

This chapter provides information about this book, including the following:

- Organization of the book for quick information retrieval
- New and enhanced DOS 4.00 services
- Contents of the utilities diskette
- Minimum hardware configuration.

Organization of this Book

This book is organized by logical application program development stages necessary to develop an application program on DOS 4.00. You must:

- Write the source code for the program
- Translate the source code to executable code
- Load and execute the program
- Debug the program
- Create object code libraries (optional)
- Convert your file formats (optional).

In addition, the book tells how to make best use of the operating system by writing your own device driver or by using the system extensions.

Each chapter describes a particular subject. You do not need to read the entire book to create programs or solve problems. Key topics also can be found by referring to the index and the table of contents.

The appendixes contain reference information for quick retrieval. They contain the entire numerical list of DOS 4.00 services, including interrupts, function calls, and device driver services.

The Technical Quick Reference contains abbreviated information about interrupts. Function calls are listed, along with input/output information and error codes.

New DOS 4.00 Services

DOS 4.00 replaces IBM DOS Version 3.30 and incorporates all services previously provided by IBM DOS Version 3.30. Major new features include:

- Enhanced country support
- Double-byte character support
- A device driver
- File system support of large disk media
- Extended Memory Support (EMS)
- Performance improvements to the file system.

Several function requests within interrupt 21H have been enhanced.

The Utilities Diskette

A utilities diskette is included with this book. It contains a listing of the following utilities, and the description of each utility, to help programmers develop an application program:

DEBUG.COM	A utility to isolate and determine errors in executable programs.
EXE2BIN.EXE	A utility to convert executable file formats (.EXE) to .COM formats to make them more compact and, therefore, load more quickly.
LIB.EXE	A utility that allows the programmer to build and edit object libraries.
LINK.EXE	A utility to translate object code to executable code.
VDISK.ASM	A fully documented programming example of a device driver. However, this example does not reflect the current level of VDISK.SYS.

Minimum Hardware Configuration

DOS 4.00 supports the following family of IBM Personal Computers and Personal System/2 computers:

The IBM Personal Computer
The IBM Personal Computer XT™
The IBM Personal Computer XT™ Model 286[1]
The IBM Personal Computer AT® (all versions)[2]
The IBM PC Convertible
The IBM Personal System/2® Model 25 (all versions)
The IBM Personal System/2® Model 30 (all versions)
The IBM Personal System/2® Model 50 (all versions)
The IBM Personal System/2® Model 60 (all versions)
The IBM Personal System/2® Model 80 (all versions).

Note: DOS 4.00 does not support the IBM PC*jr*.

The minimum memory requirement is 256KB.

---

1 Personal Computer XT is a trademark of the International Business Machines Corporation.

2 Personal Computer AT and Personal System/2 are registered trademarks of the International Business Machines Corporation.

Part 1. Writing Programs

Chapter 2. Accessing Disks

This chapter provides the necessary guide and system architecture information to help you successfully complete the following tasks:

- Accessing the disk
- Formatting the disk
- Reading and writing data to the disk.

The Disk Format

All disks and diskettes formatted by DOS 4.00 are created with a sector size of 512 bytes. DOS 4.00 is formatted on a diskette or on a designated partition of a fixed disk in the following order:

DOS 4.00 Component	Size
The boot record	1 sector
The first copy of the File Allocation Table (FAT)	Variable
The second copy of the FAT	Variable
The disk root directory	Variable
The data area	Variable

The Boot Record

The DOS 4.00 FORMAT command creates the boot record. For diskettes, the boot record resides on track 0, sector 1, side 0. While for fixed disks, it resides at the starting sector of the partition. Accessing any media (diskette or fixed disk) that does not have a valid boot record causes an error message.

The File Allocation Table (FAT)

The File Allocation Table (FAT) occupies the sectors immediately following the boot record. If the FAT is larger than one sector, the sectors occupy consecutive sector numbers.

The FAT keeps track of the physical location of all files on the disk. If the FAT cannot be read because of a disk error, the contents of the files cannot be located. For this reason, two copies of the FAT are written on the disk.

DOS 4.00 uses the FAT to allocate disk space to a file, one cluster at a time. The FAT consists of a 12-bit entry (1.5 bytes) or a 16-bit entry (2 bytes) for each cluster on the disk. On a fixed disk, the number of sectors for each cluster are determined by the size of the disk. DOS 4.00 determines whether to create a 12-bit or 16-bit FAT by calculating the number of 8-sector clusters that can occupy the space on the disk. If the number of clusters is less than 4086, a 12-bit FAT is created. If it is greater, a 16-bit FAT is created.

Using the following formula, you can determine the number of sectors on a disk:

TS = SPT * H * C.

TS = the total number of sectors on the disk.
SPT = the number of sectors per track or per cylinder.
H = the number of heads.
C = the number of cylinders.

The number of sectors on a 10MB IBM fixed disk, for example, is 20740 (17 * 4 * 305).

The first two entries in the FAT are not used to map data. They indicate the size and format of the disk. The first byte of the FAT designates one of the following:

Hex Value	Meaning
FF	Double-sided, 8 sectors per track diskette
FE	Single-sided, 8 sectors per track diskette
FD	Double-sided, 9 sectors per track diskette
FC	Single-sided, 9 sectors per track diskette
F9	Double-sided, 15 sectors per track diskette (1.2 MB)
F9	Double-sided, 9 sectors per track diskette (720 KB)
F8	Fixed disk
F0	Others

The first two FAT entries indicate the size and format of the disk. The second and third bytes of the FAT contain the value FFH. The fourth byte, used by 16-bit FATs only, contains the value FFH.

The maximum size 16-bit FAT supported by DOS 4.00 for media greater than 32MB is 64KB entries, or 128KB of space on the disk. This is an increase in size from the IBM PC DOS 3.30 limit of 16KB entries.

The Disk Directory

When the FORMAT command is issued, it builds the root directory for all disks. If the disk is formatted with the /S option, the DOS 4.00 system files, (IBMBIO.COM, IBMDOS.COM, and

COMMAND.COM), are added to the disk. The following seven formats are used for 5.25-inch diskettes and 3.5-inch diskettes:

Sides	Sectors/ Track	FAT Size Sectors	DIR Sectors	DIR Entries	Sectors/ Cluster
1 (5.25)	8	1	4	64	1
2 (5.25)	8	1	7	112	2
1 (5.25)	9	2	4	64	1
2 (5.25)	9	2	7	112	2
2 (5.25)	15	7	14	224	1
2 (3.5)	9	3	7	112	2
2 (3.5)	18	9	14	224	1

The Data Area

Data files and subdirectories are stored in the last and largest part of a disk. Space is allocated as it is needed, a cluster at a time. This allocation method permits the most efficient use of disk space. As clusters become available, space can be allocated for new files.

Accessing the Disk

Most interrupt 21H functions can be used to access a disk. Five other functions can be used to perform disk-related activity.

Activity	Function Number
Resetting the disk and flushing the file buffer	0DH
Selecting the default disk drive	0EH
Accessing the current disk	19H
Requesting the amount of free space on the disk	36H
Determining the drive at start-up time	33H

Reading and Writing Data Directly to the Disk

DOS 4.00 provides two interrupts, 25H and 26H, to read and write data to a disk.

Activity	Interrupt Number
Reading from specified disk sectors	25H
Writing to specified disk sectors	26H

Requesting Drive and Disk Information

Information on disks and drives can be requested by using the following functions:

Activity	Function Number
Requesting the current drive number	19H
Requesting disk allocation information	1BH
Requesting disk allocation information about the specified drive	1CH

Chapter 3. Accessing Files with File Handles

The information necessary to complete the following tasks is provided in this chapter:

- Reading and writing data to a file
- Requesting and specifying file attributes
- Accessing directories
- Searching for files in directories
- Requesting and specifying National Language Support (NLS).

DOS 4.00 provides nine functions within interrupt 21H to create, open, close and delete a file.

Activity	Function Number
Creating a new file or replacing an old file	3CH
Opening a file	3DH
Closing a file handle	3EH
Deleting a file	41H
Renaming a file	56H
Creating a new file with a unique name	5AH
Creating a new file	5BH
Locking and unlocking read/write access to regions of a file	5CH
Creating and opening a file with extended parameters	6CH

Filenames

To name a file, the application program supplies a pointer to an ASCIIZ string giving the name and location of the file. A filename

contains an optional drive letter, path, and/or file specification terminated with a hexadecimal 0 byte. Following is an example of a filename string:

```
'B:\LEVEL1\LEVEL2\FILE1',0
```

The maximum size of a filename is 128 bytes, including the drive, colon, path, name, and null terminator. All function calls that accept path names accept a forward slash (/) or backslash (\) as path separator characters.

File Handles

The open or create function calls return a 16-bit value called a *file handle*. To perform file I/O, a program uses the file handle to reference the file. Once a file is opened, the program no longer needs to maintain the ASCIIZ string pointing to the file. DOS 4.00 keeps track of the location of the file, regardless of which directory is current.

Activity	Function Number
Specifying an additional file handle for a file	45H
Pointing the existing file handle to another file	46H
Specifying the number of open file handles	67H

The number of file handles that can be open at one time by all processes can be specified with the FILES command in CONFIG.SYS. There are 20 default handles available to a single process. All handles inherited by a process can be redirected.

Each open handle is associated with a single file or device, but several handles can reference the same file or device. Thus, the maximum handle limit can exceed the number specified with the FILES command.

Special File Handles

DOS 4.00 provides five special file handles for use by application programs. The handles are:

0000H Standard input device (STDIN)

0001H Standard output device (STDOUT)

0002H Standard error device (STDERR)

0003H Standard auxiliary device (STDAUX)

0004H Standard printer device (STDPRN)

File handles associated with standard devices do not need to be opened by a program, but a program can close them. STDIN should be treated as a read-only file. STDOUT and STDERR should be treated as write-only files. STDIN and STDOUT can be redirected. Function calls 01H through 0CH access the standard devices.

The standard device handles are useful for performing I/O to and from the console device. For example, you can read input from the keyboard using the read function call (3FH) and file handle 0000H (STDIN); you can also write output to the console screen with the write function call (40H) and file handle 0001H (STDOUT).

If you want to prevent redirection of your output to STDOUT, you can send it using file handle 0002H (STDERR). This facility also is useful for error messages or prompts to the user.

Reading and Writing Data to a File

DOS 4.00 provides five functions to allow reading and writing to a file or device, specifying the offset within a file at which the read or write is to occur, and verifying the read-after-write state. The verification operation, however, slows performance.

Activity	Function Number
Reading from a file or device	3FH
Writing to a file or device	40H
Specifying the address (through the pointer) at which a read or write is to occur	42H
Requesting the read-after-write state	54H
Specifying the read-after-write state	2EH

Requesting and Specifying File Attributes

While a file is being created, your program can specify certain attributes; for example, the date and time of creation and level of access.

Activity	Function Number
Requesting and specifying a file's attributes	43H
Requesting and specifying a file's date and time	57H

Accessing Subdirectories

Subdirectories, that is, directories other than the root directories, are files. There is no limit to the number of subdirectory entries if the physical media can accommodate them. All directory entries are 32 bytes long.

Note: Values are in hexadecimal.

The Filename

Bytes 0 through 7 represent the filename. The first byte of the filename indicates the status of the filename. The status of a filename can contain the following values:

00H Filename never used. To improve performance, used to limit the length of directory searches.

05H The first character of the filename has an E5H character.

E5H Filename has been used, but the file has been erased.

2EH The entry is for a directory. If the second byte is also 2EH, the cluster field contains the cluster number of this directory's parent directory. (Cluster number 0000H if the parent directory is the root directory.)

Any other character is the first character of a filename.

Note: Byte offsets are in decimal.

The Filename Extension

Bytes 8 through 10 indicate the filename extension.

The File Attribute

Byte 11 indicates the file's attribute. The attribute byte is mapped as follows:

01H Indicates a read-only file. An attempt to open the file for output using function call 3DH or 6CH results in an error code being returned.

02H Indicates a hidden file. The file is excluded from normal directory searches.

04H Indicates a system file. The file is excluded from normal directory searches.

08H Indicates the entry contains the volume label in the first 11 bytes. The entry contains no other usable information and may exist only in the root directory.

10H Indicates the entry defines a subdirectory and is excluded from normal directory searches.

20H Indicates an archive bit. The bit is set ON when the file has been written to and closed. It is used by the BACKUP and RESTORE commands for determining whether the file has been changed since it was created or last updated. This bit can be used along with other attribute bits.

All other bits are reserved and must be 0.

The File Creation/Last Changed Time

Bytes 22 and 23 contain the time when the file was created or last updated. The time is mapped in the bits as follows:

```
<       23          > <         22          >
15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
h  h  h  h  h  m  m  m  m  m  m  x  x  x  x  x
```

Where:

hh = the binary number of hours (0-23)
mm = the binary number of minutes (0-59)
xx = the binary number of two-second increments

The time is stored with the least significant byte first.

The File Creation Date

Bytes 24 and 25 contain the date when the file was created or last updated. The mm/dd/yy are mapped in the bits as follows:

```
<       25          > <         24          >
15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
y  y  y  y  y  y  y  m  m  m  m  d  d  d  d  d
```

Where:

mm = 1−12
dd = 1−31
yy = 0−119 (1980−2099)

The date is stored with the least significant byte first.

The Starting Cluster Number

Bytes 26 and 27 contain the cluster number of the first cluster in the file. The first cluster for data space on all fixed disks and diskettes is cluster 002. The cluster number is stored with the least significant byte first.

```
<       27          > <         26          >
 0  0  0  0  0  0 0  0  0  0  1  0  0  0  0  1
```

The File Size

Bytes 28 through 31 contain the file size in bytes. The first word contains the low-order part of the size. Both words are stored with the least significant byte first.

Accessing Directories

DOS 4.00 provides four functions within interrupt 21H to create, move, change or delete directories.

Activity	Function Number
Removing a subdirectory	3AH
Creating a subdirectory	39H
Changing to another directory	3BH
Identifying the current directory	47H

Finding Files in Directories

DOS 4.00 provides two functions within interrupt 21H to search for the first matching entry and the next matching entry.

Activity	Function Number
Searching for the first matching entry	4EH
Searching for the next matching entry	4FH

Requesting and Specifying National Language Support (NLS)

DOS 4.00 provides the following functions for NLS:

Activity	Function Number
Specifying the current country	38H
Requesting the country dependent information	38H
Providing double-byte character set (DBCS) support	65H

Controlling Network Operations

Several DOS 4.00 function calls accept a network path as input if the IBM PC Local Area Network is loaded. If network access is available, further information is noted in the "Remarks" section under each relevant function call in Appendix B, "DOS 4.00 Function Calls" on page B-1.

A network path consists of an ASCII string containing a computer name,a directory path, and an optional filename. The network path cannot contain a drive specifier. The path is terminated by a byte of binary 0's. Following is an example:

```
\\SERVER1\LEVEL1\LEVEL2\FILE1
```

Many function calls that accept an ASCIIZ string as input accept a network path. If you want to execute function 5BH (Create a New File), for example, you must have Read/Write/Create or Write/Create access to the directory to be able to create a file. If you have Read Only or Write Only access and no Create access, you cannot create a file in the directory. Two function calls that do not accept a network path as input are Change Current Directory (3BH) and Find First Matching File (4EH).

The following function calls are available to control network operations:

Activity	Function Number
Locking and unlocking read/write access to a region of a file	5CH
Writing all data from a file to a device	68H
Requesting the local computer ID	5E00H
Specifying the printer setup string	5E02H
Requesting the printer setup string	5E03H
Requesting redirection	5F02H
Attaching to a redirect device	5F03H
Canceling redirection	5F04H

Chapter 4. Accessing Files Using File Control Blocks

This chapter provides guide and system architecture information to assist in performing the following tasks:

- Accessing files
- Accessing sequential records
- Accessing random records
- Finding files in directories
- Requesting drive and disk information.

The File Control Block (FCB)

With few exceptions, a program should maintain files using File Control Blocks (FCBs) only to run under DOS 1.10. Using FCBs, your program is restricted to the use of function calls 00H through 2EH only. File handles are the recommended method for accessing files.

One FCB maintained by your program and DOS 4.00 is required for each open file. Your program must supply a pointer to the FCB and fill in the appropriate fields required by specific function calls.

A program should not attempt to use the reserved fields in the FCB. Bytes 0 through 15 and 32 through 36 must be set by the user program. Bytes 16 through 31 are set by DOS 4.00 and must not be changed by user programs.

An unopened FCB consists of the FCB prefix (if used), the drive number, the filename, and the extensions appropriately specified. An open FCB is one in which the remaining fields have been specified by the create or open function calls.

All word fields are stored with the least significant byte first. For example, a record length of 128 is stored as 80H at offset 14, and 00H at offset 15. Figure 4-1 (The File Control Block) gives further explanation.

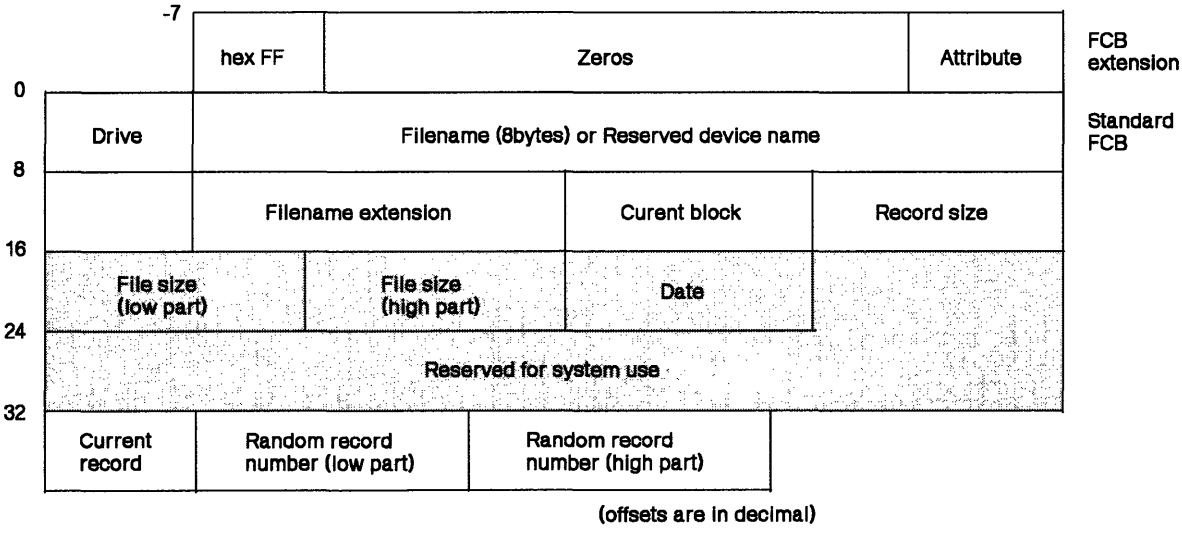


Figure 4-1. The File Control Block

Note: Offsets are in decimal.

Unshaded areas must be filled in by the using program.

Shaded areas are filled in by DOS and must not be modified.

The FCB is formatted as follows:

Drive Number

Byte 0 represents the drive number. For example, before an open 0 equals the default drive, 1 equals drive A, and 2 equals drive B. After an open 0 equals drive A, 1 equals drive A, and 2 equals drive B.

The actual drive number replaces the 0 when a file is opened.

Filename

Bytes 1 through 8 represent the filename, left-justified with trailing blanks. If a reserved device name such as LPT1 is specified here, do not include the colon.

Filename Extension

Bytes 9 through 11 represent the filename extension, left-justified with trailing blanks or all blanks.

Current Block Number

Bytes 12 through 13 represent the current block number relative to the beginning of the file, starting with 0. The 0 is set by the open function call. A block consists of 128 records, each size specified in the logical record size field. The current block number is used with the current record field for sequential reads and writes.

Logical Record Size

Bytes 14 through 15 represent the logical record size in bytes. 80H is set by the open function call. If you want to change the logical record size from 80H, you can reset the value. DOS 4.00 uses the value to determine locations in the file for all disk reads and writes.

File Size

Bytes 16 through 19 represent file size in bytes. In this two-word field, the first word is the low-order part of the size.

File Date

Bytes 20 through 21 represent the date the file was created or last updated. The *mm/dd/yy* are mapped in the bits as follows:

```
<        21        > <        20        >
15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
y  y  y  y  y  y  y  m  m  m  m  d  d  d  d  d
```

where:

mm is 1-12
dd is 1-31
yy is 0-119 (1980-2099)

Reserved

Bytes 22 through 31 are reserved.

Record Number in Block

Byte 32 represents the current relative record number (0-127) within the current block. You must set this field before doing sequential read and write operations to the diskette. This field is not initialized by the open function call.

Record Number within File

Bytes 33 through 36 represent the record number relative to the beginning of the file, starting with 0. You must set this field before doing random read and write operations to the diskette. This field is not initialized by the open function call.

If the record size is less than 64 bytes, both words are used. If the record size is more than 64 bytes, only the first 3 bytes are used. Note that if you use the FCB at 5CH in the program segment, the last byte of the FCB overlaps the first byte of the unformatted parameter area.

The Extended FCB

The extended FCB is used to create or search in the disk directory for files with special attributes. The extension adds a 7-byte prefix to the FCB, formatted as follows:

Extended FCB

FCB byte -7 contains FFH to indicate an extended FCB.

Reserved

FCB bytes -6 to -2 are reserved.

File Attribute

FCB byte -1 represents an attribute byte. Function calls 00H through 2EH are valid for both the standard FCB and the extended FCB. If you are using an extended FCB, the appropriate register should be set to the first byte of the prefix, rather than the drive number field.

The Disk Transfer Area (DTA)

DOS 4.00 uses a buffer in memory, the Disk Transfer Area (DTA), to hold the data for FCB file reads and writes. The DTA can be at any location within the data area of your program and should be specified by your program.

Only one DTA can be in effect at a time, so your program must tell DOS 4.00 which memory location to use before issuing disk read or write functions. When a program is given control by COMMAND.COM, a default DTA large enough to hold 128 bytes is established at 80H in the program segment prefix.

DOS 4.00 provides the following functions within interrupt 21H to handle DTA activities:

Activity	Function Number
Specifying the buffer address for reading and writing data in the DTA	1AH
Requesting the buffer address for reading and writing data in the DTA	2FH

Accessing Files

An FCB can identify a file on any valid drive, but only in the current directory of the specified drive.

If SHARE has not been loaded, the number of files that can be open at a time (using FCB function calls) is not restricted. When file sharing is loaded, however, the maximum number of FCB opened files is limited by the value specified in the FCBS command in CONFIG.SYS.

You can specify two values, *m* and *n*, using the FCBS command. The *m* value specifies the total number of files that can be opened by FCBS. The *n* value specifies the number of files opened by FCBs that are protected from being closed.

When the maximum number of FCB opens is exceeded, DOS 4.00 automatically closes the least recently used file. Any attempt to access such a file results in the interrupt 24H critical error message, "FCB not available." If this situation occurs while a program is running, the value specified for *m* in the FCBS command should be increased.

Do not use the same FCB to open a second file without closing the first open file. If more than one file is to be opened concurrently, use separate FCBs. To avoid potential file sharing problems, close files after I/O is performed. Close the file before trying to delete or rename an open file.

Managing files using the FCBS command can be performed using the following function calls:

Activity	Function Number
Opening a file	0FH
Closing a file	10H
Deleting a file	13H
Creating a file	16H
Renaming a file	17H
Requesting the file size	23H
Separating the filename information into its components (parsing)	29H

Accessing Sequential Records

By using the current block, current record, and record length fields of the FCB, you can perform sequential I/O by using the following sequential read or write function calls within interrupt 21H:

Activity	Function Number
Reading from a record	14H
Writing to a record	15H

Accessing Random Records

Random I/O can be performed by filling in the random record and record length fields in the FCB and issuing the following function calls within interrupt 21H:

Activity	Function Number
Reading from a single record	21H
Writing to a single record	22H
Specifying the random record field in the FCB	24H
Reading from multiple records	27H
Writing to multiple records	28H

Finding Files in Directories

Using the FCB as a source, finding and changing files in directories is performed by the following functions within interrupt 21H:

Activity	Function Number
Searching for the first matching file entry	11H
Searching for the next matching file entry	12H
Creating a file	16H
Deleting a file	13H
Renaming a file	17H
Separating the filename information into its components (parsing)	29H

Chapter 5. Managing Device I/O

This chapter provides guide and system architecture information about the following tasks:

- Managing display I/O
- Managing keyboard I/O
- Managing miscellaneous I/O
- Managing device redirection
- Accessing the system device drivers' control channel.

Managing Display I/O

DOS 4.00 provides four functions within interrupt 21H that send characters or strings of characters to the screen.

Activity	Function Number
Outputting a character to the screen, with the ability to trigger the control-break interrupt handler	02H
Waiting until a character is input and outputting it to the screen without the ability to trigger the control-break interrupt handler	06H
Outputting a string of characters in memory to the screen	09H
Outputting a string of characters in a buffer to the screen or writing the string to a file device	40H

For further information on specifying character attributes, foreground and background screen colors, and screen size using ANSI.SYS, see the *DOS 4.00 Command Reference*.

Managing Keyboard I/O

DOS 4.00 provides a full complement of functions within interrupt 21H that your application program can use to manage keyboard I/O.

Activity	Function Number
Sending input from the keyboard (with echo) to the display	01H
Receiving input directly from the keyboard, or sending output directly to the display	06H
Receiving input directly from the keyboard without echo	07H
Receiving input from the keyboard without echo to the display with the ability to trigger the control-break interrupt handler	08H
Reading characters from the keyboard to the buffer	0AH
Checking the keyboard buffer status	0BH
Clearing the keyboard buffer; specifying which function to call after clearing the buffer	0CH

For further information on reassigning the keys, see the *DOS 4.00 Command Reference*.

Managing Miscellaneous I/O

Three functions are available to manage miscellaneous I/O.

Activity	Function Number
Auxiliary Input	03H
Auxiliary output	04H
Printer output	05H

Managing File System Activities

The following system activities are supported by DOS 4.00:

Activity	Function Number
Requesting the local computer ID	5E00H
Specifying the printer setup string	5E02H
Requesting the printer setup string	5E03H
Requesting redirection list	5F02H
Attaching to a redirect device	5F03H
Canceling redirection	5F04H
Writing all data from a file to a device	68H

Accessing the System Device Drivers' Control Channel

Function 44H within interrupt 21H is a multi-purpose function for accessing the device drivers' control channel. Using function 44H, your application program can request the status of a device and read and write to the I/O control channel. The following subfunction values should be passed in AL:

Category	Activity	Subfunction Number
Requesting and specifying device information	Requesting device information	00H
	Specifying device information	01H
	Determining whether a device contains removable media	08H

Category	Activity	Subfunctl Number
Reading and writing data to a character device	Reading from a character device	02H
	Writing to a character device	03H
Reading and writing data to a block device	Reading from a block device	04H
	Writing to a block device	05H
Requesting and specifying the logical drive	Requesting the logical drive	0EH
	Specifying the logical drive	0FH
Providing network support for devices	Specifying how many times (and intervals) DOS 4.00 should try to resolve shared file conflicts	0BH
	Controlling I/O for file handles	0CH
	Controlling I/O for block devices	0DH
	Determining whether a logical device is local or remote	09H
	Determining whether a file handle is local or remote	0AH

Reading and Writing Data in Binary and ASCII Modes

A program can use function 44H to change the mode in which data is read or written to a device. If I/O is performed in binary mode, control values have no meaning. If I/O is performed in ASCII mode, certain control values have meaning. They are shown in the following table:

Control Value	Keyboard Input	Meaning
1AH	^Z	End-Of-File
0DH	^M	Carriage Return
0AH	^J	Line Feed
03H	^C	Control Break
13H	^S	Scroll Lock
10H	^P	Print Screen
11H	^Q	Scroll restart
04H	^D	End of Task

When a file is read in ASCII mode, it is echoed to the display and tabs are expanded into spaces. They are left as a tab byte (09H) in the input buffer. When a file is written in ASCII mode, tabs are expanded to 8-character boundaries and filled with spaces (20H).

Chapter 6. Controlling Processes

This chapter provides guide and system architecture information about the following activities:

- Identifying a program at load time
- Loading and executing a subprogram
- Terminating a program/subprogram
- Loading an overlay without executing it
- Calling a command processor
- Responding to errors
- Responding to a control-break action
- Requesting and specifying the system date and time
- Requesting and specifying the interrupt vectors.

Allocating Memory

DOS 4.00 keeps track of allocated and available memory blocks and provides three function calls for application programs to communicate their memory requests.

Activity	Function Number
Allocating memory	48H
Freeing allocated memory	49H
Changing the size of blocks of allocated memory	4AH

DOS 4.00 Memory Management

DOS 4.00 manages memory by allocating 16-byte units called *paragraphs* and building a *control block* for each allocated block. Any allocation is 16 bytes larger than the actual request because DOS 4.00 automatically allocates a control block to keep track of each allocated block.

When the user starts the program at the command line, COMMAND.COM loads the executable program module into the largest unused block of available memory and reads the file header.

If there is not enough memory available, the system returns an error code and passes control to the program. Your program should use the SETBLOCK function call (4AH) to reduce allocated memory to the size it needs.

Note: Because it is likely that the default stack supplied by DOS 4.00 lies in the area of memory being freed, a .COM program should remember to set up its own stack before issuing a SETBLOCK. The SETBLOCK call frees unneeded memory which then can be used for loading subsequent programs.

If your program requires additional memory during processing, issue function call 48H within interrupt 21. To free memory, issue function call 49H within interrupt 21.

The DOS 4.00 Memory Map

The following table illustrates the order in which the DOS 4.00 components and application programs are located in memory:

Location	Use
0000:0000	Interrupt vector table
0040:0000	ROM communication area
0050:0000	DOS 4.00 communication area
XXXX:0000	IBMBIO.COM – DOS 4.00 interface to ROM I/O routines
XXXX:0000	IBMDOS.COM – DOS 4.00 interrupt handlers, service routines (INT 21 functions)
XXXX:0000	DOS 4.00 buffers, control areas, and installed device drivers
XXXX:0000	Resident portion of COMMAND.COM – Interrupt handlers for interrupts 22H (terminate), 23H (Ctrl-Break), 24H (critical error), and code to reload the transient portion

Location	Use
XXXX:0000	External command or utility — .COM or .EXE file
XXXX:0000	User stack for .COM files
XXXX:0000	Transient portion of COMMAND.COM

Memory map addresses are in segment:offset format. For example, 0070:0000 is absolute address 00700H.

The DOS 4.00 Communication Area is used as follows:

0050:0000 Print screen status flag store

- 0 Print screen not active or successful print screen operation
- 1 Print screen in progress
- 255 Error encountered during print screen operation

0050:0001 Used by BASICA

0050:0004 Single-drive mode status byte

- 0 Diskette for drive A was last used
- 1 Diskette for drive B was last used

0050:0010—0021 Used by BASICA

0050:0022—002F Used by DOS 4.00 for diskette initialization

0050:0030—0033 Used by MODE command.

All other locations within the 256 bytes beginning at 0050:0000 are reserved for DOS 4.00 use.

Identifying a Program at Load Time

DOS 4.00 provides two function calls for application programs to specify and identify themselves at load time:

Activity	Function Number
Creating the means for DOS 4.00 to identify a program at load time through the program segment prefix (PSP)	26H
Requesting how DOS 4.00 identified a program at load time	62H

The Program Segment

When you enter an external command or call a program with the EXEC function call (4BH), DOS 4.00 determines the lowest available address in memory and assigns it to the program. That area of memory is called the *program segment*. At offset 0 within the program segment, DOS 4.00 builds a *program segment prefix* control block. When an EXEC is issued, DOS 4.00 loads the program at offset 100H and gives it control. See Figure 6-1 on page 6-5 for an illustration of the program segment prefix.

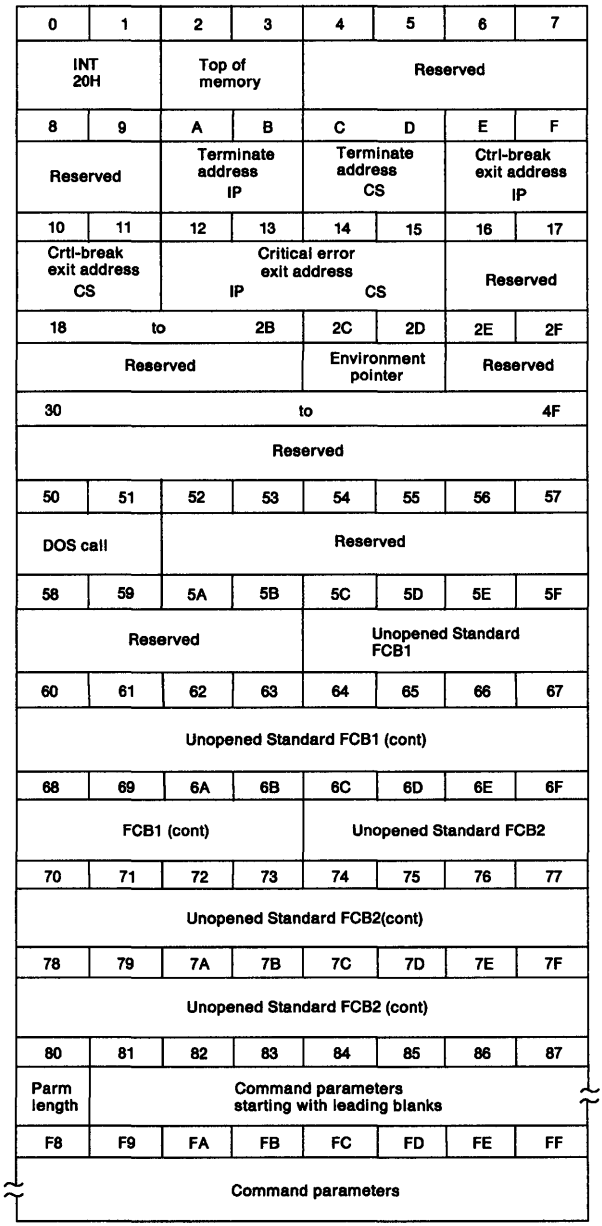


Figure 6-1. The Program Segment Prefix

The program segment prefix's first segment of available memory is in paragraph form; that is, 1000H represents 64KB. The word at offset 6 contains the number of bytes available in the segment.

Offset 2CH contains the environment's paragraph address.

Offset 50H contains code to invoke the DOS 4.00 function dispatcher. By placing the desired function number in AH, a program can issue a long call to PSP+50H to invoke a DOS 4.00 function rather than issuing an interrupt 21H.

The default disk transfer address is set to 80H.

An unformatted parameter area at 81H contains all the characters entered after the command name, including leading and imbedded delimiters, with 80H set to the number of characters. If the <, >, or | parameters were entered on the command line, they and the filenames associated with them will not appear in this area because redirection of standard input and output is transparent to applications.

For .COM files, offset 6 (one word) contains the number of bytes available in the segment.

Register AX contains the drive specifiers entered with the first two parameters as follows:

AL=FFH if the first parameter contained an invalid drive specifier (otherwise AL=00H).
AH=FFH if the second parameter contained an invalid drive specifier (otherwise AH=00H).

In .EXE programs DS and ES registers are set to point to the program segment and CS, IP, SS, and SP registers are set to the values passed by the Linker.

In .COM programs all four segment registers contain the segment address of the initial allocation block, starting with the program segment prefix control block. The instruction pointer (IP) is set to 100H. The SP register is set to the end of the program's segment. The segment size at offset 6 is rounded down to the paragraph size.

Loading and Executing Overlays

Your program can use the 4BH function call to load optional overlays. Function 4BH, value 0, loads and executes a program with overlays. Function 4BH, value 3, loads an overlay without executing it.

If your program calls an overlay, the EXEC call assumes the calling program has already allocated memory for the overlay. The request to load an overlay does not verify that the calling program owns the memory into which the overlay is to be loaded. An overlay loaded into memory not allocated to it can damage the DOS 4.00 memory management control blocks. This will not be evident until DOS 4.00 needs to use its series of control blocks.

If a memory allocation error is returned, the problem must be corrected and the system restarted. Overlays should not issue SETBLOCK calls because they do not own the memory in which they operate. The memory is controlled by the calling program.

The Parameter Block

When your program calls a subprogram using the EXEC call (4BH), it can pass a parameter block which provides the subprogram with the following:

- The environment string
- A command line which permits it to act like another command processor
- File control blocks at 5C and 6C in the program segment prefix (optional).

The Environment String

The environment passed from the calling program is a copy of its environment. The segment address of the passed environment is contained at offset 2CH in the program segment prefix.

The environment is a series of ASCII strings totaling less than 32KB in the form:

NAME=parameter

Note: NAME= is always in uppercase.

Each string is terminated by a byte of 0's. The complete series of strings is terminated by another byte of 0's. Another ASCII string containing the word count and an ASCIIZ string containing the executable program's drive, path, filename, and extension follow the series of environment strings.

The environment built by the command processor and passed to all called programs contains a COMSPEC= *string*, the last PATH, APPEND and PROMPT commands issued, and any environment strings specified with the SET command.

The Command Line

Your program must create a command line which will be transferred to the subprogram.

The File Control Blocks

If your program is using files based on file handles, the file control blocks are of no concern. If your program is using file control blocks, and either 5CH or 6CH contain a pathname, the corresponding FCB will contain only a valid drive number. The filename field will not be valid.

Terminating a Program/Subprogram

DOS 4.00 provides four functions and two interrupts to terminate programs. It also provides an interrupt to permit your program to specify where control is to be passed upon termination.

Activity	Function Number
Terminating a program and passing control to the calling process	4CH
Terminating a program with a specified portion remaining in memory	31H

Activity	Function Number
Terminating a program	00H
Determining how a process ended	4DH

Interrupt 20H terminates a program. Interrupt 27H terminates a program with a specified portion remaining in memory. Interrupt 22H specifies where control is to be passed upon program termination.

When a subprogram terminates, control is returned to the calling program. Before terminating, the calling program must return to the system the memory it allocated to the subprogram. When the calling program terminates, control is returned to DOS 4.00. DOS 4.00 does a CHECKSUM to determine if the transient portion of COMMAND.COM has been modified. If it has, DOS 4.00 reloads COMMAND.COM based on the path specified in the environment.

The program returns from executing in one of the following methods:

- By a jump to offset 0 in the program segment prefix
- By issuing an INT 20H
- By issuing an INT 21H with register AH=00H or 4CH
- By calling location 50H in the program segment prefix with AH=00H or 4CH.

  Using INT 21H is the preferred method.

  All programs must ensure that the CS register contains the segment address of the program segment prefix when terminating using any of the preceding methods except call 4CH.

  All of the preceding methods return control to the program that issued the EXEC. During the process, interrupt vectors 22H, 23H, and 24H (terminate, Ctrl-Break, and critical error exit addresses) are restored from the values saved in the program segment prefix of the terminating program. Control is then given to the terminating address.

Loading an Overlay without Executing It

If AL=3 is specified within function call 4BH, no program segment prefix is built, and DOS 4.00 assumes the calling program has allocated memory for the overlay. The calling program should provide memory in one of two ways:

- Provide enough memory for the overlay when it issues the SETBLOCK call (4AH)
- Free adequate memory with the 49H call.

When DOS 4.00 receives an AL=3 request, the system assumes that the requested memory is owned by the calling program. As in subprograms, an overlay can be loaded into memory not allocated to it and damage the series of DOS 4.00 memory management control blocks.

Programs loaded with AL=3 should not issue the SETBLOCK call (4AH) because the memory in which they operate is owned by the calling process, not the overlay. Before terminating, the calling program must return to the system the memory it allocated to the overlay. When the calling program terminates, control is returned to DOS 4.00.

Calling a Command Processor

To call a command processor, you must do the following:

- Assure that adequate free memory is available to contain the second copy of the command processor and the command it is to execute. Issue function call 4AH to shrink allocated memory to your current requirement. Issue function call 48H with BX=FFFFH. The return is available memory.
- Build a parameter string for the secondary command processor in the form:

  1 byte = length of parameter string
  xx byte = parameter string
  1 byte = 0DH (carriage return)

For example, the following assembly statement builds the string to execute a DISKCOPY command:

```
DB 19, "/C C:DISKCOPY A: B:" , 13
```

- Use the EXEC function call (4BH, function value 0) to execute the secondary copy of the command processor. The COMSPEC = parameter in the environment passed at PSP+2CH identifies the drive, directory, and command processor name. Remember to set offset 2 in the EXEC control block to point to the parameter string.

Responding to Errors

When a DOS 4.00 function cannot be performed (indicating a critical error situation) control is transferred to interrupt 24H. Function 59H provides additional information on the error condition.

Activity	Number
Responding to a critical error situation	Interrupt 24H
Requesting additional error information and suggested action	Function 59H

Handle function calls report an error by setting the carry flag and returning the error code in AX. FCB function calls report an error by returning FFH in AL.

The Extended Error function call (59H) provides a common set of error codes and specific error information such as error classification, location, and recommended action. In most critical cases, applications can analyze the error code and take specific action. Recommended actions are intended for programs that do not understand the error codes. Programs can take advantage of extended error support both from interrupt 24H critical error handlers and after issuing interrupt 21H function calls. Do not code to specific error codes.

Responding to a Control-Break Action

Interrupt 23H is issued if a Ctrl-Break occurs during standard I/O. Function calls 09H and 0AH can be used if there is a ^C, carriage return and line feed produced as output.

Activity	Function Number
Responding to a control-break action	23H
Displaying string	09H
Buffering keyboard input	0AH

If a Ctrl-Break is entered during standard input, standard output, standard printer, or asynchronous communications adapter operations, an INT 23H is executed. If BREAK is on, INT 23H is checked on most function calls, except 06H and 07H.

The user-written Ctrl-Break routine can use function calls 09H, 0AH, and 0DH to respond to the Ctrl-Break action by having ^C, carriage return, and line feed produced as output. ASCII codes 0DH and 0AH represent carriage return and line feed, respectively. If the Ctrl-Break routine saves all registers, it may end with an IRET (return from interrupt) instruction to continue program execution. If the routine returns with a long return, the carry flag is used to determine whether or not to stop execution. If the carry flag is not set, execution continues, as with an IRET.

There are no restrictions on what the Ctrl-Break handler is allowed to do, providing the registers are unchanged if IRET is used.

Requesting and Specifying the System Date and Time

The following functions get or set the system date and time:

Activity	Function Number
Requesting the system date	2AH
Specifying the system date	2BH
Requesting the system time	2CH
Specifying the system time	2DH

Requesting and Specifying the Interrupt Vectors

A program can create and change the contents of the *interrupt vectors*, the 4-byte addresses of the routines in memory that service hardware and software interrupts. On exit, the program must reset the interrupt vectors to where they were pointing originally.

If you want a program to examine or specify the contents of an interrupt vector, use DOS 4.00 function calls 35H and 25H and avoid referencing the interrupt vector locations directly.

Activity	Function Number
Requesting the interrupt vector value	35H
Specifying the interrupt vector value	25H

Part 2. Using the Programming Utilities

Chapter 7. Creating Object Code Libraries

This chapter describes how to use the IBM Library Manager/2, LIB.EXE on your utilities diskette, to create and maintain object code libraries.

The IBM Library Manager/2

The IBM Library Manager/2 allows you to store object files in library modules so they can be referred to by your application program object files.

You can provide input to the IBM Library Manager/2 (LIB) by:

- Responding to a series of prompts
- Entering input at the command line
- Using a response file you have created.

We recommend that you allow LIB to prompt you for responses until you are comfortable with its parameters and operations.

By entering information at prompts supplied by LIB, you can:

- Create a new library from one or more object files.
- Modify an existing library by:
  - Adding a module
  - Erasing a module
  - Replacing a module
  - Copying a module to an object file
  - Removing a module to an object file.
- Combine libraries.
- Create a library cross-reference listing.
- Perform a library consistency check.
- Set the library page size.

The prompts and the allowable responses are described below. See "Creating and Maintaining Libraries" on page 7-9 for a description of library management tasks.

Starting the LIB.EXE Utility

To start LIB, type

```
LIB
```

at the command line and press the Enter key. LIB prompts you for information by displaying the following prompts, one at a time:

```
Library name:
Operations:
List file:
Output library:
```

Each time LIB displays a prompt, it waits for your response before it displays the next one. With the exception of the first prompt, you can respond by just pressing the Enter key. LIB supplies the default response and takes you to the next prompt. At any prompt you can select the defaults for all the remaining prompts by typing a semicolon (;) and pressing the Enter key.

An error message will cause the LIB session to end. For a list of LIB error messages, see "Library Manager Error Messages" on page 7-15. You can cause the library manager session to end at any time by pressing Ctrl+Break. Control is then returned to DOS 4.00.

The responses to LIB prompts are described below.

Library Name Prompt

After you have entered LIB to start the library manager, the following prompt is displayed:

```
Library Name:
```

Here is where you tell LIB the name of library you want it to manage. You can type the filename of an existing library, or you can type a new filename for a library you want to create. You must enter the name of a library; there is no default response.

If you do not include an extension with your filename, LIB automatically supplies a .LIB extension.

If the library is an existing one that does not have a .LIB extension, you must type its extension; otherwise LIB will not be able to find the

library. If the existing library is in a directory other than your current one or on another drive, you must include this information.

Perform a consistency check: At this prompt you can have LIB perform a consistency check on an existing library by typing a semicolon (;) immediately after the library name and pressing Enter. This procedure is usually not necessary for libraries created with LIB.

If you are creating a library, type a new file specification. You can include a drive and path. LIB will respond by displaying this prompt:

```
Library file does not exist.  Create?
```

To confirm you want to create a library, type **Y**. Typing **N** ends the LIB session and returns you to the DOS prompt.

Set the page size: At this prompt you can specify a page size for the library. Type the page size option **/P**:*n*, where *n* is the number of bytes in a page. If you do not specify a page size, the default is 16 bytes for a new library and the current page size if the library already exists. See "Setting the Library Page Size" on page 7-13 for more information.

Operations Prompt

LIB displays the next prompt:

```
Operations:
```

At this prompt you can type any one of the command symbols described below, immediately followed by the name of an object file, a library, or a library module, depending on the operation you are performing. When you use the name of a module, remember that a module name has no path and no extension.

You can include as many operations (a symbol followed by the name being operated on) as will fit on the line.

When you are manipulating a large number of modules or files, you can type more than one line of information by typing an ampersand (&) as the last character on the line and pressing Enter. LIB will then repeat the **Operations** prompt, allowing you to enter more information. The ampersand must follow a filename and not a command symbol.

The following table lists the symbols used at the **Operations** prompt:

Symbol and Task Description		Example
+	*Add an object file to a library:* Type a plus sign followed by an object filename to copy the contents of the file into a library module. If the object file is not in your current directory, specify a path. If you omit the .OBJ extension, LIB looks for a file with an .OBJ extension.	+c:\obj1
+	*Add a library:* Type a plus sign followed by a library filename to copy a library to the library specified at the **Library name** prompt: You must type the library file's extension. Otherwise LIB looks for a file with an .OBJ extension.	+mylib.lib
–	*Erase a module:* Type a minus sign followed by a module name to erase the module from the library. (Remember that a module name has no path and no extension.)	–moda
–+	*Replace a module:* Type a minus sign, a plus sign, and a module name to replace the module with the contents of an object file of the same name. LIB assumes the object file is in your current directory and has an .OBJ extension.	–+modb
*	*Copy a module:* Type an asterisk (*) followed by a module name to copy the module to an object file of the same name. The module remains in the library. LIB gives the new file an .OBJ extension and places it in your current working directory.	*modc
–*	*Remove a module:* Type a minus sign, asterisk, and module name to remove the module from the library and place it in an object file of the same name. LIB places the new file in your current directory and gives it an .OBJ extension.	–*moda

The default for the **Operations** prompt is no operations performed.

List File Prompt

This prompt is displayed:

```
List File:
```

Type a filename at this prompt if you want to create a cross-reference listing for your library. You can include a drive and path. If you do not include an extension, LIB does not supply a default extension.

If you do not type a filename, the default filename is NUL.LST, which means cross-reference information will not be saved.

For a description of the contents of the cross-reference listing, see "Creating a Cross-Reference Listing" on page 7-12.

Output Library Prompt

This prompt is displayed:

```
Output library:
```

Next to the prompt is the name of the library you specified at the **Library name** prompt.

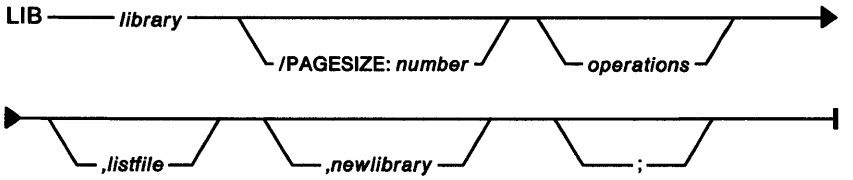
If you typed in the name of an existing library at the **Library name** prompt, and you want to create a new library with the changes you have made, type in a new name.

If you typed in the name of a new library at the **Library name** prompt, and you want to change the name, type in a new name.

If you accept the default and press the Enter key, your library is given the name you typed at the **Library name** prompt. If the library is an existing one, the original library keeps the same name but is given an extension of .BAK.

Entering Input at the Command Line

To supply input to LIB at the command line, enter the parameters in the format that follows. You may include a drive and path specification for any of the filenames you enter.



library This required parameter along with the optional /PAGESIZE*number* parameter corresponds to the **Library name** prompt. If you want LIB to perform a consistency check on the library, type a semicolon after the *library* name.

/PAGESIZE This parameter sets the library page size in bytes. The minimum abbreviation you can type is /P:*number*. The allowable values for *number* are 16, 32, 64, 128, 256, or 32,768. For more information on the /PAGESIZE option, see "Setting the Library Page Size" on page 7-13.

operations This parameter corresponds to the **Operations** prompt. If you are typing in a lot of operations and know that your input will exceed a line boundary, type an ampersand (&) after the last complete operation on the line and press the Enter key. LIB then displays the **Operations** prompt so you can continue typing.

See "Operations Prompt" on page 7-3 for descriptions of the operations you can enter.

,listfile This parameter corresponds to the **List file** prompt. It tells LIB to create a cross-reference listing for your library. If you specify a filename, you must type a comma preceding the filename. See "Creating a Cross-Reference Listing" on page 7-12 for a description of this file's contents.

If you do not want a list file created, specify an additional comma instead of a filename.

,newlibrary This parameter corresponds to the **Output Library** prompt. Specify a filename if you want a new name for a revised library. You must type a comma preceding the filename. If you do not want to specify a filename here, specify an additional comma instead.

; A semicolon tells LIB you have completed your input.

If you enter a partial command line without a semicolon, LIB displays the prompts for the remaining parameters. You can enter a value as each prompt appears or press the Enter key to accept the default. If you type a semicolon after any parameter, LIB supplies the defaults for the remaining parameters.

Examples of Command Line Input

To add the file TEST.OBJ to the library BASIC.LIB without producing a cross-reference, type:

```
LIB BASIC.LIB+TEST.OBJ;
```

Note that the following has the same result as the preceding example:

```
LIB BASIC.LIB+TEST;
```

Extensions are optional, and they default to .OBJ if omitted. If you are using a library file in an operations list, you must specify the .LIB extension.

To erase TEST from BASIC.LIB, type:

```
LIB BASIC-TEST;
```

To replace TEST in the library with a newer version, type:

```
LIB BASIC-+TEST;
```

Note that the following also have the same effect:

```
LIB BASIC-TEST+TEST.OBJ;

LIB BASIC+TEST-TEST;
```

If you want to make the same change, except to put the changes in a new library called BASNEW.LIB, any of the following will work:

```
LIB BASIC-+TEST,,BASNEW

LIB BASIC-TEST+TEST,,BASNEW

LIB BASIC+TEST-TEST,,BASNEW
```

If you want to create a library of object modules, type:

```
LIB MYSUBS+FILE1.OBJ+FILE2.OBJ+...+FILEN.OBJ
```

You are asked for the listing file.

Using a Response File

To operate the library manager with a response file, you must first create a file that contains the responses you want LIB to process. You can give your file any name you want. The responses you put in the file must be in the same order as the LIB prompts:

```
Library name
Operations
List file
Output library name
```

The library name and the list of operations can appear on one line. If your operations extend over one line, end the line with an ampersand (&) and continue on the next line.

You can omit responses in your file and instead provide these responses with a partial command line or have LIB prompt you for them.

Once you have completed typing the responses, end the file with either a semicolon or a carriage return and line feed combination. If you do not end your file, LIB will display the last line of the file and wait for you to press the Enter key.

When you are ready to tell LIB to process the contents of your response file, you must specify the file with a preceding **at** sign (@), like this:

```
LIB @C:\ABC\MYFILE.RSP
```

This tells LIB the file specification is for a response *file*, and not a response to a prompt. Remember to include a path with your file specification when the file is not in the current directory.

When you use the LIB command with a response file, LIB displays each prompt on your screen with the corresponding response from your file.

You can have LIB process only a part of your response file by placing a semicolon on any line in the response file. When LIB reads the semicolon, it supplies defaults for the remaining responses and ignores the rest of the response file.

You can place all of your responses to the prompts in a response file. The library manager is instructed by the following example to read the responses from the response file:

```
LIB @RESP.TXT
```

where RESP.TXT is the filename containing the responses.

Note: RESP.TXT was chosen as an example. Any valid DOS filename can be used.

Creating and Maintaining Libraries

This section summarizes the library management tasks you can perform with LIB.

Creating a Library File

To create a new library file, type the name of the library file you want to create following the **Library name** prompt. LIB supplies the .LIB extension.

If the name of the new library is the name of an existing file, LIB assumes you want to modify the existing file. When you give the name of a library file that does not currently exist, LIB displays the following prompt:

```
Library file does not exist.  Create?
```

Type **y** (yes) to create the file; or **n** (no) to end the library session.

Note: When you call LIB in such a way that no **Operations** prompt appears, the message above does not appear. LIB assumes y (create the new library) by default. For example,

```
LIB new.lib+obj1;
```

where **new.lib** does not exist, it creates the file **new.lib**.

You can specify a page size for the library when you create it. The default page size is 16 bytes. See "Setting the Library Page Size" on page 7-13 for more information.

After you give the name of the new library file, you can insert object modules in the library by using the add operation (+) following the **Operations** prompt. You can also add the contents of another library. See "Adding Library Modules" on page 7-11 and "Combining Libraries" on page 7-12 for an explanation of these options.

Modifying a Library File

You can change an existing library file by giving the name of the library file following the **Library name** prompt. The **Operations** prompt performs all operations you specify on that library.

LIB lets you keep both the original library file and the changed version. You can do this by giving the name of a new library file following the **Output library** prompt. The library filename changes to the new library filename, while the original library file remains unchanged.

If you do not give a filename following the **Output library** prompt, the changed version of the library file replaces the original library file. LIB saves the original, unchanged library file. The original library file has the extension .BAK instead of .LIB. At the end of the session you have two library files: the changed version with the .LIB extension and the original, unchanged version with the .BAK extension.

Adding Library Modules

Use the plus sign following the **Operations** prompt to add an object module to a library. Give the name of the object file, without the .OBJ extension, that you want to add immediately after the plus sign.

LIB removes the drive designation and the extension from the object file specification, leaving only the filename. This becomes the name of the object module in the library. For example, if the object file B:\CURSOR.OBJ is added to a library file, the name of the corresponding object module is CURSOR. LIB always adds object modules to the end of a library file.

Deleting Library Modules

Use the minus sign following the **Operations** prompt to delete an object module from a library. Give the name of the module you want to delete immediately after the minus sign. A module name has no pathname and no extension. It is only a word, such as CURSOR.

Replacing Library Modules

Use a minus sign followed by a plus sign to replace a module in the library. After the replacement symbol (-+), give the name of the module you want to replace. Module names have no pathnames and no extensions.

To replace a module, LIB deletes the given module, and adds the object file with the same name as the module. The object file has an .OBJ extension and resides in the current working directory.

Extracting Library Modules

Use an asterisk (*) followed by a module name to copy a module from the library file into an object file of the same name. The module also remains in the library file. When LIB copies the module to an object file, it adds the .OBJ extension, the drive designation, and the pathname of the current working directory to the module name. This forms a complete object filename. You cannot override the .OBJ extension, drive designation, or pathname given to the object file. You can later rename the file or copy it to another location.

Use the minus sign followed by an asterisk (-*) to move an object module from the library file to an object file. This operation is equivalent to copying the module to an object file, then deleting the module from the library.

Combining Libraries

You can add the contents of a library to another library by using the plus sign with a library filename instead of an object filename. Following the **Operations** prompt, give the plus sign and the name of the library you want to add to the one you are changing. When you use this option, you must include the .LIB extension of the library filename. Otherwise, LIB assumes that the file is an object file and looks for the file with an .OBJ extension.

LIB adds the modules of the library to the end of the library you are changing. The added library still exists as an independent library. LIB copies the modules without deleting them.

After you add the contents of a library or libraries, you can save the new, combined library under a new name by giving a new name following the **Output library** prompt. If you omit the **Output library** response, LIB saves the combined library under the name of the original library you are changing.

Creating a Cross-Reference Listing

You can create a cross-reference listing by giving a name for the listing file following the **List file** prompt. If you omit the response to this prompt, LIB uses the special filename NUL.LST, which tells LIB not to save the cross-reference information.

A cross-reference listing file contains two lists. The first is an alphabetic listing of all public symbols in the library. The name of the module the symbol name refers to comes after that symbol name. For example, the cross-reference for BASCOM20.LIB contains:

```
$SWPA.......SWAP        $SWPB.......SWAP
$SWPC.......SWAP        $SWPD.......SWAP
```

In this case, the module names are all the same.

The second list is an alphabetic list of the modules in the library. Under each module name is an alphabetic listing of the public

symbols to which the module refers. The module name includes its offset value, and its code and data size in hexadecimal. In this example, the module's name is SWAP.

```
SWAP Offset: 21970H Code and data size: 4FH
$SWPA       $SWPB        $SWPC       $SWPD
```

Performing Consistency Checks

When you give only a library name followed by a semicolon at the **Library name** prompt, LIB performs a consistency check, displaying messages about any errors it finds. It does not make any changes to the library. This option is not usually necessary because LIB checks object files for consistency before adding them to the library.

To produce a cross-reference listing along with a consistency check, use the command prompt method of calling LIB. Give the library name followed by a comma, then give the name of the listing file. LIB performs the consistency check and creates the cross-reference listing.

Setting the Library Page Size

The page size of a library affects the alignment of modules stored in the library. Modules in the library are aligned so that they always start at a position that is a multiple of *n* bytes from the beginning of the file. The value of *n* is the page size. The default page size is 16 for a new library or the current page size for an existing library.

Because of the indexing technique LIB uses, a library with a larger page size can hold more modules than a library with a smaller page size. However, for each module in the library, this indexing technique wastes an average of *n*/2 bytes of storage space (where *n* is the page size). In most cases a small page size is advantageous. You should use the smallest page size possible.

To set the library page size, add a page size option after the library filename in response to the **Library name** prompt:

library-name **/PAGESIZE:***n*

The value of *n* is the new page size. It must be a power of 2 and must be between 16 and 32768.

Another consequence of this indexing technique is that the page size determines the maximum possible size of the .LIB file. This limit is 65536 times *number*. For example, **/P:16** means that the .LIB file must be smaller than 1 megabyte (16 times 65536 bytes) in size.

Library Manager Error Messages

Error messages produced by the IBM Library Manager/2, LIB, have one of the following formats:

- *filename*|LIB : fatal error U1*xxx* : *messagetext*
- *filename*|LIB : error U2*xxx* : *messagetext*
- *filename*|LIB : warning U4*xxx* : *messagetext*

The message begins with the input filename (*filename*), if one exists, or with the name of the utility. LIB may display the following error messages:

U1150 page size too small
The page size of an input library is too small, which indicates a non-valid input .LIB file.

U1151 syntax error : illegal file specification
You gave a command operator, such as a minus sign, without a module name following it; or an illegal filename or no filename at all.

U1152 syntax error : option name missing
You gave a forward slash (/) without a value following it.

U1153 syntax error : option value missing
You gave the **/PAGESIZE** option without a value following it.

U1154 option unknown
An unknown option is given. Currently, LIB recognizes the **/PAGESIZE** option only.

U1155 syntax error : illegal input
The given command did not follow correct LIB syntax.

U1156 syntax error
The given command did not follow correct LIB syntax.

U1157 comma or new line missing
A comma or carriage return is expected in the command line, but did not appear. This may indicate an inappropriately placed comma, as in the following line:

LIB math.lib,-mod1 + mod2;

The line should have been entered as follows:

LIB math.lib -mod1 + mod2;

U1158 terminator missing
Either the response to the **Output library:** prompt or the last line of the response file used to start LIB did not end with a carriage return.

U1161 cannot rename old library
LIB could not rename the old library to have a .BAK extension because the .BAK version already existed with read-only protection. Change the protection of the old .BAK version.

U1162 cannot reopen library
The old library could not be reopened after it was renamed to have a .BAK extension.

U1163 error writing to cross-reference file
The disk or root directory is full. Delete or move files to make space.

U1170 too many symbols
More than 4609 symbols appeared in the library file.

U1171 insufficient memory
LIB did not have enough memory to run. Remove any shells or resident programs and try again, or add more memory.

U1172 no more virtual memory
You should note the conditions when the error occurs and contact your authorized IBM dealer.

U1173 internal failure
You should note the conditions when the error occurs and contact your authorized IBM dealer.

U1174 mark : not allocated
You should note the conditions when the error occurs and contact your authorized IBM dealer.

U1175 free : not allocated
You should note the conditions when the error occurs and contact your authorized IBM dealer.

U1180 write to extract file failed
The disk or root directory is full. Delete or move files to make space.

U1181 write to library file failed
The disk or root directory is full. Delete or move files to make space.

U1182 *filename* **: cannot create extract file**
The disk or root directory is full, or the specified extract file already exists with read-only protection. Make space on the disk or change the protection of the extract file.

U1183 cannot open response file
The response file was not found.

U1184 unexpected end-of-file on command input
An end-of-file character is received prematurely in response to a prompt.

U1185 cannot create new library
The disk or root directory is full, and the library file already exists with read-only protection. Make space on the disk or change the protection of the library file.

U1186 error writing to new library
The disk or root directory is full. Delete or move files to make space.

U1187 cannot open VM.TMP
The disk or root directory is full. Delete or move files to make space.

U1188 cannot write to VM
You should note the conditions when the error occurs and contact your authorized IBM dealer.

U1189 cannot read from VM
You should note the conditions when the error occurs and contact your authorized IBM dealer.

U1190 interrupted by user
User typed control-break while LIB was executing.

U1200 *name* **: invalid library header**
The input library file has an invalid format. Either it is not a library file, or it has been corrupted.

U1203 *name* **: invalid object module near** *location*
The module specified by *name* is not a valid object module.

U2152 *filename* **: cannot create listing**
The directory or disk is full, or the cross-reference listing file already exists with read-only protection. Make space on the disk or change the protection of the cross-reference listing file.

U2155 *modulename* **: module not in library; ignored**
The specified module is not found in the input library.

U2157 *filename* **: cannot access file**
LIB is unable to open the specified file.

U2158 *libraryname* **: invalid library header; file ignored**
The input library has an incorrect format.

U2159 *filename* **: invalid format** *hexnumber*; **file ignored**
The signature byte or word, *hexnumber*, of an input file is not one of the recognized types.

U4150 *modulename* **: module redefinition ignored**
A module is specified to be added to a library, but a module with the same name is already in the library. Or a module with the same name is found more than once in the library.

U4151 *symbol(modulename)* **: symbol redefinition ignored**
The specified symbol is defined in more than one module.

U4153 *number* **: page size invalid; ignored**
The value specified in the **/PAGESIZE** option is less than 16.

U4156 *libraryname* **: output-library specification ignored**
An output library is specified in addition to a new library name. For example, specifying:

LIB new.lib + one.obj,new.lst,new.lib

new.lib does not exist and causes this error.

Chapter 8. Creating an Executable File

This chapter describes how to use the IBM Linker/2, which is LINK.EXE on your utilities diskette, to produce executable programs.

The IBM Linker/2

The IBM Linker/2 combines object files with referenced library subroutines to produce a file which can be relocated and executed. The object files you supply to the linker for processing must be compiled or assembled source files that are written in the programming languages supported by DOS 4.00.

You can provide input to LINK by:

- Responding to a series of prompts
- Entering input at the command line
- Using a response file you have created.

We recommend that you allow LINK to prompt you for responses until you are comfortable with its parameters and operations.

Starting the LINK.EXE Program

To start the linker, type

```
LINK
```

at the command line and press Enter. The linker prompts you for information by displaying the following prompts, one at a time:

```
Object modules [.OBJ]:
Run file [filename.EXE]:
List file [NUL.MAP]:
Libraries [.LIB]:
Definitions File [NUL.DEF]:
```

Each time the linker displays a prompt, it waits for your response before it displays the next one. With the exception of the first prompt, you can respond by pressing the Enter key. The linker supplies the default response and takes you to the next prompt. At any prompt

you can select the defaults for all the remaining prompts by typing a semicolon (;) and pressing the Enter key.

You can also type in one or more linker options at any of the prompts. A linker option instructs the linker to perform a special function as it processes your object files. For example, specifying the /EXEPACK option tells LINK to compress the .EXE file being created so it will load faster. For descriptions of the options provided by LINK and guidelines for using them, see "Using Linker Options" on page 8-11.

Any error message generated by a response to a prompt will cause the linker session to end. For a list of LINK error messages, see "Linker Error Messages" on page 8-37. You can cause the linker session to end at any time by simultaneously pressing the Ctrl and Break key so Control is then returned to DOS 4.00.

Object Modules Prompt

After you enter LINK to start the linker, this prompt is displayed:

```
Object Modules [.OBJ]:
```

Here is where you enter the names of the object files to be used to create your .EXE file. You must enter the name of at least one object file. Separate each object filename from the next by a blank space or a plus sign (+). If a plus sign is the last character on the line when Enter is pressed, the Object Modules prompt reappears, allowing you to add more filenames.

If you do not type an extension for a filename, LINK automatically supplies the .OBJ extension.

Include a drive and path with your file specification if the object file is not on the current drive or in the current directory. If LINK cannot find an object file, it does not display an error message until you have answered all the prompts.

When you have completed typing your input, press the Enter key.

Run File Prompt

The next prompt is displayed:

```
Run File [filename.EXE]:
```

Here is where you name the .EXE file that will be created. The *filename* displayed in this prompt is the first one you typed next to the Object Modules prompt. If you do not supply a new name and just press the Enter key, this will be the name of your file. It will have an extension of .EXE.

List File Prompt

The next prompt is displayed:

```
List File [NUL.MAP]:
```

The linker list file is sometimes called the linker *map*. It contains the names, load addresses, and lengths of all segments in a program. It also lists the names and load addresses of any groups in the program, the program's starting address, and messages generated by any errors the linker may have encountered. For more information on map file contents, see "Reading the Map File" on page 8-31.

If you do not want a list file, do not type a filename. When you press the Enter key, the file NUL.MAP is created, which means LINK will not save mapping information.

If you enter a filename without an extension, LINK supplies .MAP.

By typing the /MAP option after your file specification, you tell LINK to include a list of all public symbols and their addresses in the map file. If you enter only /MAP without a filename, LINK creates a map file that includes the public symbol information and has the same name as the first object file you specified for the Object Modules prompt. Its extension will be .MAP.

Libraries Prompt

The next prompt is displayed:

```
Libraries [.LIB]:
```

List the names of the libraries you want linked and the names of any directories you want searched for library names whose paths have not been specified. Separate these specifications with a blank space or a plus sign (+).

When you supply directory specifications, LINK uses them to search for default compiler libraries referred to in object files. If you do not supply a filename extension for a library, LINK substitutes .LIB.

The linker uses the environment variable LIB to search for libraries. Using the SET command, you can assign the names of directories as well as complete file specifications to LIB as search paths. If you have defined search paths to the linker with LIB, this determines what information you need to enter. For example, if you have included the search path to the directory that contains your library in the value you set for LIB, you need enter only the library name, not its path, at the prompt. If you have included the entire file specification in the LIB definition, then this library will be searched when references to public symbols are encountered. For a description of the SET command, refer to the *DOS 4.00 Command Reference.*

The maximum number of search paths you can enter at this prompt is limited to 32. If your input is going to exceed a line, make the last character on the line a plus sign (+) and press the Enter key. LINK then repeats the prompt so you can continue to type.

To locate the default libraries, LINK searches in the following order:

1. The current working directory
2. The directories in the order listed following the **Libraries** prompt
3. The libraries specified by the LIB environment variable.

LINK searches all the libraries until it finds the first definition of a library public symbol.

If you do not require any libraries to be searched for object code, press the Enter key.

Definition File Prompt

This is the prompt displayed by LINK:

```
Definition File [.DEF]:
```

Since this IBM Linker/2 parameter for creating a module definition file is an IBM Operating System/2 parameter that is not supported by DOS 4.00, press the Enter key in response to this prompt.

This completes your input to LINK, and LINK will now process the information you have supplied. If no errors are encountered, LINK creates your executable file.

LINK uses available memory for the link session. If the files to be linked create an output file that exceeds available memory, LINK creates a temporary disk file to serve as storage. LINK creates the temporary file in the current working directory. If you are working in a directory that is on a diskette, LINK displays the following message:

```
Temporary file name has been created.

Do not change diskette in drive letter:
```

The *name* is a unique temporary filename created by the linker. After this message appears, do not remove the diskette from the given drive (*letter*) until the link session ends. If you remove the diskette, the operation of LINK is unpredictable. If LINK unexpectedly ends, you may see the following message:

```
Unexpected end of file on name
```

If you get this message, you must restart link from the beginning. After LINK creates the executable file, it automatically deletes the temporary file.

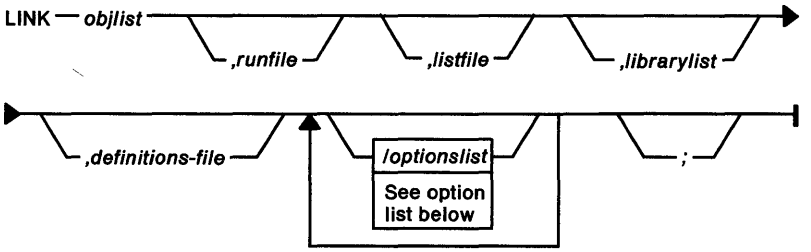
An Example of a Prompts Session

The following example links the object modules MODA.OBJ, MODB.OBJ, MODC.OBJ, and STARTUP.OBJ. LINK searches the library file MATH.LIB in the \LIB directory on drive B for routines and data used in the programs. LINK then creates an executable file named MODA.EXE and a map file named ABC.MAP. The **/PAUSE** option in the **Object Modules** prompt line causes LINK to pause while you change diskettes. LINK then creates the executable file.

```
LINK

Object Modules [.OBJ]: moda + modb +
Object Modules [.OBJ]: modc + startup /PAUSE
Run File [MODA.EXE]:
List File [NUL.MAP]:abc
Libraries [.LIB]: b:\lib\math
Definitions File [NUL.DEF]:
```

Entering LINK Input at the Command Line

To run the linker from the command line, type the responses to LINK prompts as parameters to the LINK command in the format that follows.



With the exception of linker options, which begin with a forward slash (/) and can appear anywhere on the command line, parameters specified to LINK must be typed in the same order as the prompts are presented by LINK. Separate parameters by commas, and if you omit a parameter within a sequence, specify an additional comma.

If you enter a partial command line that does not end in a semicolon, LINK displays the prompts for the remaining parameters. You can enter values for these prompts or just press the Enter key to accept the defaults.

Refer to "Starting the LINK.EXE Program" on page 8-1 for default descriptions.

objlist This parameter corresponds to the **Object Files** prompt. The linker requires at least one object file. Separate file specifications by a blank space or a plus sign (+). If your input will exceed one line, type a plus sign as the last character on the line before pressing Enter. The linker then displays the **Object Files** prompt.

,runfile This parameter corresponds to the **Run File** prompt. If specified, this is the name of the executable file created by LINK.

,listfile This parameter corresponds to the **List File** prompt. If specified, this is the name of the map file.

,librarylist This parameter corresponds to the **Libraries** prompt. If specified, this is the list of libraries LINK searches for object code routines.

,definitions file This parameter for creating module definition files corresponds to the **Definitions File** prompt. Module definition files are not supported by DOS 4.00.

/optionslist This parameter represents any linker options you wish to specify. For the purpose of simplicity, the command format shows this position for linker options. Actually, you can put linker options anywhere on the command line. For a description of linker options, see "Using Linker Options" on page 8-11.

; The semicolon tells LINK you have completed your input. LINK supplies the defaults for any remaining parameters.

Examples of Command Line Input

In the example below, an object module FILE.OBJ is processed to create the executable file FILE.EXE. The linker searches the library ROUTINE.LIB for routines and variables used within the program. It also creates a file called FILE.MAP, which contains a list of the segments of the program and groups. The following examples creates the file:

```
LINK file.obj,file.exe,file.map,routine.lib;
```

It is equal to the following line:

```
LINK file , ,file, routine;
```

In the following example, the linker loads and links the object modules FUN.OBJ, TEXT.OBJ, TABLE.OBJ, and CARE.OBJ and searches for unresolved references in the library file COBLIB.LIB. By default, the executable file produced is named FUN.EXE. A list file named FUNLIST.MAP is also produced.

```
LINK FUN+TEXT+TABLE+CARE,,FUNLIST,COBLIB.LIB /MAP;
```

In the next example, LINK uses the two object modules STARTUP.OBJ and FILE.OBJ on the current drive to create an executable file named FILE.EXE on drive B:. LINK also creates a map file in

the **\MAP** directory of the current drive but does not search any libraries.

```
LINK startup+file,b:file,\map\file;
```

To link the application object file SAMPLE.OBJ using the the libraries LIB1.LIB and LIB2.LIB, type:

```
LINK sample,sample.exe, sample.map/M/LI, lib1+lib2/NOD;
```

This command creates the file SAMPLE.EXE. It also creates the map file SAMPLE.MAP. The command searches the library files LIB1.LIB and LIB2.LIB to resolve any external references made in SAMPLE.OBJ. The **/NOD** option directs LINK to ignore any default libraries specified in the object file.

The linker uses default filename extensions if you do not explicitly provide your own. In the example above, LINK extends the first occurrence of the filename SAMPLE to SAMPLE.OBJ. LINK extends the library files with the .LIB extension. The **/LI** option has LINK copy the line number information from the object files.

Using a Response File to Supply LINK Input

To operate the linker with a response file, you must first create a file that contains the responses you want LINK to process. You can give your file any name you want. The responses you put in the file must be in the same order as the LINK prompts:

objectfiles
[*runfile*]
[*listfile*]
[*libraryfiles*]
[*definitionsfile*]

Each response to a prompt must begin on a separate line, but you can extend long responses across more than one line by typing a plus sign as the last character of each incomplete line. You can place one or more options on any line.

You can omit responses in your file and instead provide these responses with a partial command line or have LINK prompt you for them.

Once you have completed typing the responses, end the file with either a semicolon or a carriage return and line feed combination. If you do not end your file, LINK will display the last line of the file and wait for you to press the Enter key.

When you are ready to tell LINK to process the contents of your response file, you must specify the file with a preceding **at** sign (@), like this:

```
LINK @C:\ABC\MYFILE.RSP
```

This tells LINK the file specification is for a response *file*, and not a response to a prompt. Remember to include a path with your file specification when the file is not in the current directory.

When you use the LINK command with a response file, LINK displays each prompt on your screen with the corresponding response from your file. If the response file does not contain responses for all the prompts, LINK displays the appropriate prompts and waits for you to enter responses. When you type an acceptable response, LINK continues the session.

You can have LINK process only a part of your response file by placing a semicolon on any line in the response file. When LINK reads the semicolon, it supplies defaults for the remaining responses and ignores the rest of the response file. For example, if you type a semicolon on the line of the response file corresponding to the **Run File** prompt, LINK uses the defaults for the rest of the responses, starting with the executable file response.

Examples of Response Files

Suppose you enter these lines in a response file:

```
FUN TEXT TABLE CARE
/PAUSE /MAP
FUNLIST
COBLIB.LIB;
```

When the response file is processed, LINK loads the four object files named FUN, TEXT, TABLE, and CARE. LINK produces two output files named FUN.EXE and FUNLIST.MAP. The **/PAUSE** option causes

LINK to pause before producing the executable file (FUN.EXE). This permits you to change diskettes if necessary. See the **/PAUSE** and **/MAP** options under "Using Linker Options" on page 8-11 for more information.

The response file below tells the linker to link the four object modules MODA, MODB, MODC, and STARTUP. The linker pauses for you to change diskettes before producing the runfile MODA.EXE. The linker also creates a map file ABC.MAP and searches the library MATH.LIB in the \LIB directory of drive B:.

```
moda modb modc startup /PAUSE

abc
b:\lib\math
```

The following example shows you how you can combine all three methods of supplying file names. Assume that you have a response file called LIBRARY that contains the following line:

```
lib1+lib2+lib3+lib4;
```

Now start LINK with a partial command line:

```
LINK object1 object2
```

LINK takes OBJECT1.OBJ and OBJECT2.OBJ as its object files, and asks for the next line with the following:

```
Run File [object1.EXE]:  exec
List File [NUL.MAP]:
Libraries [.LIB]:  @library
```

Type **exec** so that the linker names the run file EXEC.EXE. Press the Enter key to show that you do not want a map file, and type **@ library** for the linker to use in the response file containing the four library filenames.

Using Linker Options

The table below lists LINK options and describes what they do.

You can type linker options at any prompt or at any point on the command line. You can specify an option's minimum abbreviation as shown in the **Options** column, or as many characters of its long form as you like. The long forms appear in parentheses. Remember to include the leading forward slash (/) for each option you specify.

A complete format description, usage guidelines, and examples for each option follow this table.

Option	Task Description
/CO	Prepare the .EXE file for later symbolic debugging (/CODEVIEW). You should not use this option with the /EXEPACK option.
/CP	Reduce the maximum amount of paragraphs reserved for your program from the default of 64KB minus 1 to whatever you specify (/CPARMAXALLOC).
/DO	Force the ordering of segments (/DOSSEG).
/DS	Load data starting at the high end of the segment (/DSALLOCATE). This option is used with the /HIGH option.
/E	Compress the .EXE file for faster loading (/EXEPACK). You should not use this option with the /CODEVIEW option.
/HE	Display a list of the LINK options on your screen (/HELP).
/HI	Cause the .EXE file to be placed in the highest storage available when loaded (/HIGH). This option is used with the /DSALLOCATE option.

Option	Task Description
/I	Display LINK processing messages (/INFORMATION).
/LI	Add a list of the starting addresses of program source lines to your map file (/LINENUMBERS). If you do not specify a map file, this option causes one to be created.
/M	Add a list of public symbols declared by your object files to your map file (/MAP). If you do not specify a map file, this option causes one to be created.
/NOD	Ignore any default compiler library names found in an object file (/NODEFAULTLIBRARYSEARCH). This allows you to supply your own library names.
/NOG	Provide compatibility with previous compiler versions (/NOGROUPASSOCIATION).
NOI	Distinguish between uppercase and lowercase characters (/NOIGNORECASE). This option is for files created by compilers that make this distinction.
/O	Set the interrupt number to other than 3FH for passing control to overlays (/OVERLAYINTERRUPT).
/PAU	Pause before generating the .EXE file (/PAUSE). This option allows you to insert a diskette.
/SE	Set the maximum number of logical segments to a value other than the the default of 128 (/SEGMENT).
/ST	Set the stack size to other than the size calculated by LINK (/STACK).

This option directs LINK to include symbolic debugging information for CodeView in the output .EXE file for IBM languages that support the CodeView debugger.

Format

/CODEVIEW

The minimum abbreviation is **/CO.**

Comments

Warning: If /CODEVIEW is used with the /EXEPACK option, all symbolic debugging information will be lost.

Reserving Paragraph Space /CPARMAXALLOC

This option allows you to change the default value of the MAXALLOC field, which controls the maximum number of paragraphs reserved in storage for your program. A paragraph is defined as the smallest storage unit (16 bytes) addressable by a segment register.

Format

/CPARMAXALLOC:*number*

The minimum abbreviation is **/CP**.

Comments

The maximum number of paragraphs reserved for a program is determined by the value of the MAXALLOC field at offset 0CH in the EXE header.

The default for the MAXALLOC field is 65535 (decimal), or 64KB minus 1. You can reset the default to any number between 1 and 65535 (decimal, octal, or hexadecimal). Reducing the number is helpful because:

- Program efficiency is not increased by reserving all available memory.
- You may need to run another program within your program and you need to reserve space for the run program.

If the value you specify is less than the computed value of MINALLOC (at offset 0AH), the linker uses the value of MINALLOC instead.

This option forces segments to be ordered according to the following rules:

1. All segments with a class name ending in CODE
2. All other segments outside of DGROUP
3. DGROUP segments in the following order:
    a. Any segments of class BEGDATA. (This class name is reserved for IBM use.)
    b. Any segments not of class BEGDATA, BSS, or STACK.
    c. Segments of class BSS.
    d. Segments of class STACK.

Format

/DOSSEG

The minimum abbreviation is **/DO**.

Controlling Data Loading
/DSALLOCATE

By default, LINK loads all data starting at the low end of the data segment. At run time, LINK sets the DS (data segment) pointer to the lowest possible address to allow the entire data segment to be used.

You can use the **/DSALLOCATE** option to tell LINK to load all data starting at the high end of the data segment. To do this, at run time, set the DS pointer to the lowest data segment address that contains program data.

Format

/DSALLOCATE

The minimum abbreviation is **/DS**.

Comments

This option is typically used with the **/HIGH** option to take advantage of unused storage within the data segment. You can reserve any available storage below the area specifically reserved for DGROUP, using the same **DS** pointer.

Packing Executable Files /EXEPACK

This option directs LINK to remove sequences of repeated bytes (typically nulls) and to optimize the load-time relocation table before creating the DOS 4.00 executable file.

Format

/EXEPACK

The minimum abbreviation is **/E**.

Comments

Executable files linked with this option are usually smaller and load faster than files linked without this option. However, you cannot use symbolic debugging programs with packed files.

This option does not always save a significant amount of disk space and sometimes actually may increase file size. Programs that have a large number of load-time relocations (about 500 or more) or long streams of repeated characters are usually shorter if packed.

Warning: If the /CODEVIEW option is used with /EXEPACK, all symbolic debugging information will be lost.

Example

This example creates a packed version of file PROGRAM.EXE.

```
LINK  program  /E;
```

Viewing the Options List /HELP

This option causes LINK to write a list of the available options to the screen. This may be convenient if you need a reminder of the available options. Do not give a filename when using the **/HELP** option.

Format

/HELP

The minimum abbreviation is **/HE**.

Example

```
LINK /HELP
```

Note: Many LINK options that are used only with IBM Operating System/2™[1] are displayed when using the /HELP option. These options cannot be used when creating programs to run on DOS 4.00.

[1] Operating System/2 is a trademark of the International Business Machines Corporation.

You can place the run file as low or as high in storage as possible. Using the **/HIGH** option directs LINK to cause the loader to place the run file as high as possible in storage without overlaying the transient portion of COMMAND.COM. The COMMAND.COM file occupies the highest area of storage when loaded. Without the **/HIGH** option, the loader places the run file as low as possible in storage.

Use the **/HIGH** option in association with the **/DSALLOCATE** option.

Format

/HIGH

The minimum abbreviation is **/HI**.

Displaying LINK-Time Information /INFORMATION

This option causes the linker to display the phase of processing it is running, and the name of each input module as it is linked. This is useful during debugging.

Format

/INFORMATION

The minimum abbreviation is **/I.**

Copying Line Numbers to the Map File /LINENUMBERS

This option directs the linker to copy the starting address of each program source line to a map file. The starting address is the address of the first instruction that corresponds to the source line.

Format

/LINENUMBERS

The minimum abbreviation is **/LI**.

Comments

LINK copies the line number data only if you give a map filename in the LINK command line, and only if the given object file has line number information. Line numbering is available in some high-level languages.

If an object file has no line number information, the linker ignores the **/LINENUMBERS** option.

Note: If you do not specify a map file in a LINK command, you can still use the **/LINENUMBERS** option to force the linker to create a map file. Place the option at or before the **List File** prompt. LINK gives the forced map file the same filename as the first object file specified in the command and gives it the default extension .MAP.

Example

This example causes the line number information in the object file FILE.OBJ to be copied to the map file FILE.MAP.

```
LINK file/LINENUMBERS,,file,mylib;
```

Producing a Public Symbol Map /MAP

This option causes LINK to produce a listing of all public symbols declared in your program. This list is copied to the map file created by the linker.

Format

/MAP:*number*

The minimum abbreviation is **/M**.

For a complete description of the listing file format, see "Reading the Map File" on page 8-31.

Comments

The *number* parameter specifies the maximum number of public symbols that the linker can sort in the map file. The limit is 32768, and the default is 2048 if no number is specified. If the limit is exceeded, the linker produces an un-sorted list and issues the following warning:

MAP symbol limit too high

If you get this error, link again with a lower number. The limit varies according to how many segments the program has and how much memory is available.

Specifying a number also causes the public symbols to be sorted by address only, not by name, regardless of the number. To reduce the size of your map files by removing the list sorted by name, link with /MAP followed by the lowest possible number that will accommodate the number of public symbols in your program.

Note: If you do not specify a map file in a LINK command, you can use the **/MAP** option to force the linker to create a map file. LINK gives the forced map file the same name as the first object file specified in the command and the default extension .MAP.

Ignoring Default Libraries /NODEFAULTLIBRARYSEARCH

This option directs the linker to ignore any library names it may find in an object file. A high-level language compiler may add a library name to an object file to ensure that a default set of libraries is linked with the program. Using this option bypasses these default libraries and lets you name the libraries you want by including them on the LINK command line.

Format

/NODEFAULTLIBRARYSEARCH

The minimum abbreviation is **/NOD**.

Example

This example links the object files STARTUP.OBJ and FILE.OBJ with routines from the libraries EM, SLIBFP, and SLIBC. Any default libraries that may have been named in STARTUP.OBJ or FILE.OBJ are ignored.

```
LINK startup+file/NOD,,,em+slibfp+slibc;
```

Preserving Compatibility /NOGROUPASSOCIATION

This option causes the linker to process a certain class of fix-up routines in a manner compatible with previous versions of the linker. This option is provided primarily for compatibility with previous versions of other IBM language compilers.

Format

/NOGROUPASSOCIATION

The minimum abbreviation is **/NOG**.

Preserving Lowercase /NOIGNORECASE

This option directs LINK to treat uppercase and lowercase letters in symbol names as distinct letters. Normally, LINK considers uppercase and lowercase letters to be identical, treating the names *TWO*, *Two*, and *two* as the same. When you use the **/NOIGNORECASE** option, the linker treats *TWO*, *Two*, and *two* as different names.

Format

/NOIGNORECASE

The minimum abbreviation is **/NOI**.

Comments

This option is typically used with object files created by high-level language compilers. Some compilers treat uppercase and lowercase letters as distinct letters and assume that the linker does the same.

Example

This command causes the linker to treat uppercase and lowercase letters in symbol names as distinct letters. The object file FILE.OBJ is linked with routines from the library \SLIBC.LIB located in the \LIB directory.

```
LINK file/NOI,,,\lib\slibc;
```

Setting the Overlay Interrupt /OVERLAYINTERRUPT

By default, the DOS 4.00 interrupt number used for passing control to overlays is 3FH. This option allows you to select a different interrupt number.

Format

/OVERLAYINTERRUPT:*number*

The minimum abbreviation is **/O**.

Comments

The *number* can be a decimal number from 0 through 255, an octal number from 0 through 0377, or a hexadecimal number from 0 to FFH. Numbers that conflict with DOS 4.00 interrupts are not prohibited, but IBM does not recommend their use.

Pausing to Change Disks
/PAUSE

This option causes LINK to pause so that you can change disks before writing the executable file to disk.

Format

/PAUSE

The minimum abbreviation is **/PAU.**

Comments

If you choose the **/PAUSE** option, the linker displays the following message before creating the run file:

About to generate .EXE file
Change diskette in drive *letter* **and press <Enter>**

The *letter* is the proper drive name. This message appears after the linker has read data from the object and library files, and after writing data to the map file, if one was specified. LINK resumes processing when you press the Enter key. After LINK writes the executable file to disk, the following message appears:

Please replace original diskette
in drive *letter* **and press <Enter>**

Note: Do not remove the disk used for the temporary file, if one has been created. If the temporary disk message appears when you have specified the **/PAUSE** option, you should simultaneously press the Ctrl and C keys to end the LINK session. Rearrange your files so that LINK can write the temporary file and the executable file to the same disk, then try again.

Pausing to Change Disks /PAUSE

Example

This command causes the linker to pause just before creating the executable file file.exe. After creating the executable file, LINK pauses again to let you replace the original disk.

```
LINK file/PAUSE,file,,\lib\math;
```

Setting the Maximum Number of Segments /SEGMENTS

This option directs the linker to process no more than *number* + *3* logical segments per program. If it meets more than this limit, the linker displays an error message and stops linking. The **/SEGMENTS** option bypasses the default limit of 128 logical segments.

Format

/SEGMENTS:*number*

The minimum abbreviation is **/SE**.

Comments

If you do not specify /SEGMENTS, the linker reserves enough storage space to process up to 128 logical segments. If your program has more than 128 logical segments, you must set the segment limit higher to increase the number of logical segments LINK can process. Set the segment limit lower if you get the following LINK error message:

Requested segment limit too high

The *number* can be any integer value in the range 1 to 3072. It must be a decimal, octal, or hexadecimal number. Octal numbers must have a leading zero. Hexadecimal numbers must start with a leading zero followed by a lowercase *x*; for example, 0x4B.

Example

This example sets the logical segment limit to 192:

```
LINK file/SE:192;
```

The following example sets the logical segment limit to 255 (X'FF'):

```
LINK moda+modb,run/SEGMENTS:0xff,ab,em+mlibfp;
```

Setting the Stack Size /STACK

This option sets the program stack to the number of bytes given by *size*. The linker automatically calculates the stack size of a program, basing the size on the size of any stack segments given in the object files. If you specify /STACK, the linker uses the given *size* in place of any value it may have calculated.

Format

/STACK:*size*

The minimum abbreviation is **/ST**.

Comments

The *size* can be any positive integer value in the range 1 through 65534. The value can be a decimal, octal, or hexadecimal number. Octal numbers must begin with a zero. Hexadecimal numbers must begin with a leading zero followed by a lowercase *x*; for example, 0x1B.

The stack size can also be changed after linking with the EXEMOD utility.

Examples

This example sets the stack size to 512 bytes:

```
LINK file/STACK:512;
```

The following example sets the stack size to 255 (X'FF') bytes:

```
LINK moda+modb,run/ST:0xFF,ab.\lib\start;
```

The following example sets the stack size to 24 (30 octal) bytes:

```
LINK startup+file/ST:030;
```

Reading the Map File

The map file lists the names, load addresses, and lengths of all segments in a program. It also lists the names and load addresses of any groups in the program, the program's start address, and messages about any errors the linker may have encountered. If you use the **/MAP** LINK option, the map file lists the names and load addresses of all public symbols.

In the map file, segment information has the general form:

```
Start   Stop     Length   Name              Class

00000H  02B86H   02B87H   _TEXT             CODE
02B90H  05091H   02502H   EMULATOR_TEXT     CODE
05092H  05092H   00000H   C_ETEXT           ENDCODE
050A0H  0520FH   00170H   EMULATOR_DATA     FAR_DATA
05210H  05245H   00036H   NULL              BEGDATA
05246H  05761H   0051CH   _DATA             DATA
```

The Start column shows the address of the first byte in the segment. The number shown is the offset from the beginning of the program. This number is referred to as the frame number. The Stop column shows the address of the last byte in the segment. The Length is the length of the segment in bytes. The Name column shows the name of the segment and the Class column shows the class name of the segment.

Group information has the general form:

```
Origin          Group

0521:0          DGROUP

Program entry point at 0000:02A0
```

If you have specified the **/MAP** LINK option, the linker adds a public symbol list to the map file. Symbols are listed twice: first in alphabetical order, then in the order of their load addresses. This list has the general form shown in the following example. (For each symbol address, the number to the left of the colon represents a frame number.)

```
Address         Publics by Name

0003:071A       $i8_implicit_exp
0003:0718       $i8_inpbas
0001:287C       $i8_input
0003:0719       $i8_input_ws
0001:0CF6       $i8_output

Address         Publics by Value

0001:0010       _main
0001:01B0       _countwords
0001:0238       _analyze
0001:02A0       __astart
0001:0368       __cintDIV
```

Creating an Overlaid Version of Your Program

You can direct LINK to create an *overlaid* version of your program. This means that the COMMAND.COM loader will load parts of your program only when they are needed, and these parts will share the same space in storage. Your overlaid program should run in less storage, but probably it will run more slowly because time is needed to read and load the code into storage.

The use of overlays is restricted to object modules built with IBM high-level languages that support overlays. Refer to the language manuals that came with your compiler for a description of overlays.

Control is passed to overlay modules by a standard 8086-long (32-bit) CALL/RETURN instruction. You cannot use long jumps or indirect calls (through a function pointer) to pass control to an overlay. When a pointer calls a function, the called function must be either in the same overlay or in the root.

Specifying an Overlay Structure to LINK

You specify the overlay structure to the linker in response to the **Object Modules** prompt. Loading is automatic. You specify the overlays in the list of modules that you submit to the linker by enclosing them in parentheses. Each parenthetical list represents one overlay. For example:

```
Object Modules [.OBJ] : a+(b+c)+(e+f)+g+(i)
```

The elements (b+c), (e+f), and (i) are overlays. The remaining modules, and any drawn from the run time libraries, make up the resident or root part of your program. LINK loads your program or root overlays into the same region of storage, so only one can be resident at a time. LINK does not allow duplicate names in different overlays, so each module can occur only once in a program.

The linker replaces calls from the root to an overlay and calls from an overlay to another overlay with an interrupt, followed by the module identifier and offset. The DOS interrupt number used by LINK is 3FH. You can change this interrupt number by specifying the /OVERLAYINTERRUPT option. See "Using Linker Options" on page 8-11 for more information.

LINK adds the name for the overlays to the .EXE file, and encodes the name of this file into the program so the overlay manager can get access to it.

How LINK Formats the .EXE File

This section is provided if you need to know how LINK uses IBM Macro Assembler/2™ pseudo-up instructions to determine the format of the .EXE file. Unlike high-level language compilers such as IBM C/2 Compiler™,[2] the IBM Macro Assembler does not check to determine if the 64KB physical segment limit of DOS 4.00 has been exceeded.

Ordering Segments

LINK copies segments to the executable file in the same order that it meets them in the object files. This order is maintained throughout the program unless the linker finds two or more segments having the same *class*, as defined by the SEGMENT pseudo-op instruction. Segments having identical classes are copied to the executable files as contiguous blocks. The linker uses a segment's *align-type* defined in the SEGMENT pseudo-op instruction to set the starting address for the segment. The align types are BYTE, WORD, PARA, and PAGE.

---

[2] Macro Assembler/2 and C/2 Compiler are trademarks of the International Business Machines Corporation.

These correspond to starting addresses at byte, word, paragraph, and page boundaries, representing addresses that are multiples of 1, 2, 16, and 256, respectively. The default *align-type* is PARA.

When the linker encounters a segment, it checks the *align-type* before copying the segment to the executable file. If the *align-type* is WORD, PARA, or PAGE, the linker checks the executable image to see if the last byte copied ends at an appropriate boundary. If not, LINK adds extra null bytes to the image.

Segment Combine-Types

LINK uses *combine-types* that are defined in SEGMENT pseudo-op instructions to determine whether two or more segments sharing the same segment name should be one segment. The *combine-types* are PUBLIC, STACK, COMMON, and PRIVATE.

- If a segment is *combine-type* PUBLIC, the linker combines it with any segments of the same name and class. When LINK combines segments, it makes the segments contiguous in storage; you can reach each address in the segments using an offset from one frame address. The result is the same as if the segments were defined as a whole in the source file.

  The linker preserves the *align-type* defined in the SEGMENT pseudo-op instruction for *each* code and data segment it adds to the combined segment.

- If a segment is *combine-type* STACK, the linker combines individual segments as it does for PUBLIC *combine-types*. For STACK segments, LINK copies an initial stack-pointer value to the executable file. This stack-pointer value is the offset to the end of the first STACK segment (or combined STACK segment) that LINK meets. If you use the STACK type for STACK segments, you need not give instructions to load the segment into the **SS** register.

- If a segment is *combine-type* COMMON, the linker combines it with any segments of the same name and class. When LINK combines COMMON segments, it places the start of each segment at the same address. This creates a series of overlapping segments. The resulting combination segment has a length equal to the longest individual segment.

- LINK assigns a default *combine-type* of PRIVATE to any segments with no explicit *combine-type* definition in the source file. LINK does not combine PRIVATE segments.

If LINK tries to combine segments that total more than 64KB, it displays an error message.

Groups

The GROUP pseudo-op instruction gives addressability to non-contiguous segments of various classes relative to the same frame address. When LINK encounters a GROUP instruction, it adjusts all storage references to items in the group to the same frame address.

Segments of a group need not be contiguous, belong to one class, or have the same *combine-type*. However, all segments specified by the GROUP must fit within 64KB of storage. If the group is smaller than 64KB of storage, LINK may place segments that are not part of the group in the same storage area.

LINK does not specifically check that all segments in a group fit within 64KB of storage. If the segments are larger than the 64KB maximum, the linker can produce a **fix-up overflow** error.

Instruction and Data Reference Errors

Once the linker knows the starting address of each segment in a program and establishes all segment combinations and groups, it attempts to fix up any unresolved references to labels and variables. The linker computes an appropriate offset and segment address and replaces the temporary address values with the new values.

The size of the value that LINK computes depends on the type of reference. If LINK discovers an error in the anticipated size of the reference, it displays a fixup overflow error message. This happens, for example, when a program tries to use a 16-bit offset to address an instruction in a segment that has a different frame address. It also occurs when the segments in a group do not fit within a single, 64K block of storage.

LINK resolves four types of references:

- Short
- Near self-relative
- Near segment-relative
- Long.
  - A **short** reference occurs in JMP instructions that try to pass control to labeled instructions in the same segment or group. The target instruction must be no longer than 128 bytes from the point of reference. The linker computes a signed, 8-bit number for this **short** reference. It displays an error message if the target instruction belongs to a different segment or group (different frame address). The linker also displays an error message if the distance from the frame address to the target is more than 128 bytes in either direction.
  - A **near self-relative** reference occurs in instructions which access data relative to the same segment or group. The linker computes a 16-bit offset for this **near self-relative** reference. It displays an error message if the data resides in more than one segment or group.
  - A **near segment-relative** reference occurs in instructions that attempt to access data either in a specified segment or group or relative to a specified segment register. LINK computes a 16-bit offset for this **near segment-relative** reference. It displays an error message if the offset of the target within the specified frame is greater than 64KB or less than 0 bytes. LINK also displays an error message if LINK cannot address the beginning of the (canonical) frame of the target.
  - A **long** reference occurs in CALL instructions that try to get access to an instruction in another segment or group. LINK computes a 16-bit frame address and a 16-bit offset for this **long** reference. The linker displays an error message if the computed offset is greater than 64K or less than 0 bytes. The linker also displays an error message if LINK cannot address the beginning of the (canonical) frame of the target.

Linker Error Messages

This section lists error messages produced by the IBM Linker.

Fatal errors cause the linker to stop running. Fatal error messages have the following format:

location: **fatal error L1***xxx*: *message text*

Non-fatal errors indicate problems in the executable file. LINK produces the executable file (and sets the error bit in the header if for protected mode). Non-fatal error messages have the following format:

location: **error L2** *xxx*: *message text*

Warnings indicate possible problems in the executable file. LINK produces the executable file (it does not set the error bit in the header if for protected mode). Warnings have the following format:

location: **error L4***xxx*: *message text*

In these messages, *location* is the input file associated with the error, or LINK if there is no input file.

If the input file is an .OBJ or .LIB file and has a module name, the module name is enclosed in parentheses, as shown in the following examples:

SLIBC.LIB(_file)
MAIN.OBJ(main.c)
TEXT.OBJ

The following error messages may appear when you link object files with LINK:

L1001 *option* **: option name ambiguous**

A unique option name does not appear after the option indicator (/). For example, the command

LINK /N main;

produces this error, since LINK cannot tell which of the three options beginning with the letter **N** is intended.

L1002 *option* **: unrecognized option name**
An unrecognized character followed the option indicator (/), as in the following example:

```
LINK /ABCDEF main;
```

L1003 *option* **: MAP symbol limit too high**
The specified symbol limit value following the MAP option is greater than 32768, or there is not enough memory to increase the limit to the requested value.

L1004 *option* **: invalid numeric value**
An incorrect value appeared for one of the linker options. For example, a character string is entered for an option that requires a numeric value.

L1006 *option* **: stack size exceeds 65534 bytes**
The size you specified for the stack in the /STACK option of the LINK command is more than 65534 bytes.

L1007 *option* **: interrupt number exceeds 255**
You gave a number greater than 255 as a value for the /OVERLAYINTERRUPT option.

L1008 *option* **: segment limit too high**
The specified limit on the /SEGMENTS option is greater than 3072.

L1009 *option* **: CPARMAXALLOC : illegal value**
The number you specified in the /CPARMAXALLOC option is not in the range 1 to 65535.

L1020 no object modules specified
You did not specify any object-file names to the linker.

L1021 cannot nest response files
A response file occurs within a response file, which DOS 4.00 does not permit.

L1022 response line too long
A line in a response file is longer than 127 characters.

L1023 terminated by user
You entered **Ctrl + C**.

L1024 nested right parentheses

You typed the contents of an overlay incorrectly on the command line.

L1025 nested left parentheses

You typed the contents of an overlay incorrectly on the command line.

L1026 unmatched right parenthesis

A left parenthesis is missing from the contents specification of an overlay on the command line.

L1027 unmatched left parenthesis

A right parenthesis is missing from the contents specification of an overlay on the command line.

L1041 resident-name table overflow

The total length of all resident names, plus three bytes per name, is greater than 65534.

L1042 nonresident-name table overflow

The total length of all nonresident names, plus three bytes per name, is greater than 65534.

L1043 relocation table overflow

There are more than 65536 load-time relocations for a single segment.

L1045 too many TYPDEF records

An object module contains more than 255 TYPDEF records. These records describe communal variables. This error can only appear with programs produced by compilers that support communal variables.

L1046 too many external symbols in one module

An object module specifies more than the limit of 1023 external symbols. You need to break the module into smaller parts.

L1047 too many group, segment, and class names in one module

The program contains too many group, segment, and class names. You need to reduce the number of groups, segments, or classes, and recreate the object files.

L1048 too many segments in one module

An object module has more than 255 segments. Split the module or combine segments.

L1049 too many segments

The program has more than the maximum number of segments. The SEGMENTS option specifies the maximum allowed number; the default is 128. Relink using the /SEGMENTS option with an appropriate number of segments.

L1050 too many groups in one module

The linker found more than 21 group definitions (GRPDEF) in a single module. Reduce the number of group definitions or split the module.

L1051 too many groups

The program defines more than 20 groups, not counting DGROUP. Reduce the number of groups.

L1052 too many libraries

An attempt is made to link with more than 32 libraries. Combine libraries, or use modules that require fewer libraries.

L1053 symbol table overflow

The program has more than 256K bytes of symbolic information, (such as public, external, segment, group, class, and file names). Combine modules or segments and recreate the object files. Eliminate as many public symbols as possible.

L1054 out of memory: reduce # in /SEGMENTS: # or /MAP: #

The linker does not have enough memory to allocate tables describing the number of segments requested. (The default is 128 or the value specified with the /SEGMENTS option.) Try linking again using the /SEGMENTS option to select a smaller number of segments (for example, use 64 if the default was used previously), or free some memory by eliminating resident programs or shells.

L1056 too many overlays

The program defines more than 63 overlays.

L1057 data record too large

A LEDATA record (in an object module) contained more than 1024 bytes of data. This is a translator (compiler or assembler) error. Note which translator (compiler or assembler) produced the incorrect object module and the circumstances, and contact your authorized IBM dealer.

L1070 segment size exceeds 64K

A single segment contains more than 64KB of code or data. Try compiling, or assembling, and linking using the large model.

L1071 segment _TEXT larger than 65520 bytes

This error is likely to occur only in small-model C programs, but it can occur when any program with a segment named _TEXT is linked using the /DOSSEG option of the LINK command. Small-model C programs must reserve code addresses 0 and 1; this is increased to 16 for alignment purposes.

L1072 common area longer than 65536 bytes

The program has more than 64KB of communal variables. This error cannot appear with object files produced by the IBM Macro Assembler/2. It occurs only with programs or other compilers that support communal variables.

L1073 file-segment limit exceeded

There are more than 255 physical or file segments.

L1074 *name* : group larger than 64KB

A group contained segments which total more than 65536 bytes.

L1075 entry table larger than 65535 bytes

You have exceeded a linker table size limit because of too many entry names. Reduce the number of names in the modules you are linking.

L1080 cannot open list file

The disk or the root directory is full, or an invalid filename was entered. Delete or move files to make space, or link again with a valid filename.

L1081 out of space for run file

The disk on which .EXE file is being written is full.
Free more space on the disk and restart the linker.

L1083 cannot open run file

The disk or the root directory is full. Delete or move files to make space.

L1084 cannot create temporary file

The disk or root directory is full. Free more space in the directory and restart the linker.

L1085 cannot open temporary file

The disk or the root directory is full. Delete or move files to make space.

L1086 scratch file missing
Internal error. You should note the conditions when the error occurs and contact your authorized IBM Computer dealer.

L1087 unexpected end-of-file on scratch file
The disk with the temporary linker-output file is removed.

L1088 out of space for list file
The disk on which the listing file is being written is full. Free more space on the disk and restart the linker.

L1089 *filename* **: cannot open response file**
The linker could not find the specified response file. This usually indicates a typing error. Correct the error.

L1090 cannot reopen list file
The original disk is not replaced at the prompt. Restart the linker.

L1091 unexpected end-of-file on library
The disk containing the library probably was removed. Replace the disk containing the library and run the linker again.

L1093 *filename*: **object file not found**
The linker could not find the specified object file.

L1101 invalid object module
One of the object modules is not valid.
If the error persists after recompiling, contact your authorized IBM dealer.

L1102 unexpected end-of-file
An invalid format for a library was found.

L1103 attempt to access data outside segment bounds
A data record in an object module specified data extending beyond the end of a segment. This is a translator error. Note which translator (compiler or assembler) produced the incorrect object module and the circumstances, and contact your authorized IBM dealer.

L1104 *filename* **: not valid library**
The specified file is not a valid library file. This error causes the linker to stop running.

L1113 unresolved COMDEF; internal error
You should note the conditions when the error occurs and contact your authorized IBM dealer.

L1114 file not suitable for /EXEPACK; relink without

For the linked program, the size of the packed load image plus the packing overhead is larger than that of the unpacked load image. Relink without the EXEPACK option.

L2001 fixup(s) without data

A FIXUP record occurred without a data record immediately preceding it. This is probably a compiler error.

L2002 fixup overflow *number* **in frame seg** *segname* **target seg** *segname* **target offset** *number*

The following conditions can cause this error:

- A group is larger than 64KB.
- The program contains an intersegment short jump or intersegment short call.
- The name of a data item in the program conflicts with that of a subroutine in a library included in the link.
- An EXTRN declaration in an assembler-language source file appeared inside the body of a segment.

This example may cause this error:

```
code    SEGMENT public  'CODE'
        EXTRN   main:far
start   PROC    far
        call    main
        ret
start   ENDP
code    ENDS
```

The following construction is preferred:

```
        EXTRN   main:far
code    SEGMENT public  'CODE'
start   PROC    far
        call    main
        ret
start   ENDP
code    ENDS
```

Revise the source file and recreate the object file.

L2003 Intersegment self-relative fixup

An intersegment self-relative fixup is not allowed.

L2004 LOBYTE-type fixup overflow

A LOBYTE fixup produced an address overflow.

L2005 fixup type unsupported

A fixup type occurred that is not supported by the linker. This is (probably) a compiler error. You should note the conditions when the error occurs and contact your authorized IBM dealer.

L2010 too many fixups in LIDATA record

There are more fixups applying to a LIDATA record than will fit in the linker's 1024-byte buffer. The buffer is divided between the data in the LIDATA record and run-time relocation items, which are eight bytes apiece, so the maximum varies from 0 to 128. This is probably a compiler error.

L2011 *name* **: NEAR/HUGE conflict**

Conflicting NEAR and HUGE attributes are given for a communal variable. This error can occur only with programs produced by compilers that support communal variables.

L2012 *name* **: array-element size mismatch**

A far communal array is declared with two or more different array-element sizes (for example, an array declared once as an array of characters and once as an array of real numbers). This error cannot occur with object files produced by the IBM Macro Assembler/2. It occurs only with compilers that support far communal arrays.

L2013 LIDATA record too large

A LIDATA record in an object module contains more than 512 bytes of data. Most likely, an assembly module contains a very complex structure definition or a series of deeply-nested DUP operators. For example, the following structure definition causes this error:

```
alpha   DB     10DUP(11 DUP(12 DUP(13 DUP(...))))
```

Simplify the structure definition and reassemble. (LIDATA is a DOS term.)

L2024 *name* **: symbol already defined**

One of the special overlay symbols required for overlay support is defined by an object.

L2025 *name* **: symbol defined more than once**

Remove the extra symbol definition from the object file.

L2028 automatic data segment plus heap exceed 64K

The size of DGROUP near data plus requested heap size is greater than 64K.

L2029 unresolved externals

One or more symbols are declared to be external in one or more modules, but they are not publicly defined in any of the modules or libraries.

A list of the unresolved external references appears after the message, as shown in the following example:

```
_exit in file(s)
 main.obj (main.c)
_fopen in files(s)
 fileio.obj(fileio.c) main.obj(main.c)
```

The name that comes before **in file(s)** is the unresolved external symbol. On the next line is a list of object modules that have made references to this symbol. This message and the list are also written to the map file, if one exists.

L2030 starting address not code (use class 'CODE')

You specified to the linker a starting address which is a segment that is not a CODE segment. Reclassify the segment to CODE, or correct the starting point.

L4001 frame-relative fixup, frame ignored

A fixup occurred with a frame segment different from the target segment where either the frame or the target segment is not absolute.

L4002 frame-relative absolute fixup

A fixup occurred with a frame segment different from the target segment where both frame and target segments were absolute.

L4012 load-high disables EXEPACK

You must select either the **/HIGH** or **EXEPACK** option. They cannot be used together.

L4014 invalid option for old-format executable file ignored

If a DOS 4.00 format program is produced, the options **/ALIGNMENT**, **/NOFARCALLTRANSLATION**, and **/PACKCODE** are meaningless, and the linker ignores them.

L4020 *name* **: code-segment size exceeds 65500**

Code segments of length 65501 through 65536 may be unreliable on the 80286 processor.

L4021 no stack segment

The program does not contain a stack segment defined with STACK combine type. This message should not appear for modules whose source code was compiled by the high-level language compilers supported by DOS 4.00, but it could appear for an assembler-language module. Normally, every program should have a stack segment with the combine type specified as STACK. You can ignore this message if you have a specific reason for not defining a stack, or for defining one without the STACK combine type.

L4022 *name1, name2* **: groups overlap**

Two groups are defined so that one starts in the middle of another. This may occur if you defined segments in an assembly file and did not correctly order the segments by class.

L4028 *name* **: segment already defined**

A segment is defined more than once with the same name. Segments must have unique names for the linker. Only the first definition of a name is recognized.

L4029 *name* **: DGROUP segment converted to type data**

A segment which is a member of DGROUP is defined as type CODE in an object file.

L4030 *name* **: segment attributes changed to conform with automatic data segment**

The segment named *name* is defined in DGROUP, but the *shared* attribute is in conflict with the *instance* attribute. For example, the *shared* attribute is NONSHARED and the *instance* is SINGLE, or the *shared* attribute is SHARED and the *instance* attribute is MULTIPLE. The bad segment is forced to have the right *shared* attribute and the link continues. The image is not marked as having errors.

L4031 *name* **: segment declared in more than one group**

A segment is declared to be a member of two different groups. Correct the source file and recreate the object files.

L4032 *name* **: code-group size exceeds 65500 bytes**

Code segments of length 65501 through 65536 may be unreliable on the 80286 processor.

L4034 more than 239 overlay segments; extra put in root

You specified an overlay structure containing more than 239 segments. The extra segments have been assigned to the root overlay.

L4036 no automatic data segment

No group named DGROUP is declared.

L4050 too many public symbols

The **/MAP** option is used to request a sorted listing of public symbols in the map file, but there were too many symbols to sort. (The default is 2048 symbols.) The linker produces an unsorted listing of the public symbols. Relink using **/MAP**:*number*.

L4051 *filename* **: cannot find library**

The linker could not find the specified file. Enter a new filename, a new path specification, or both.

L4053 VM.TMP : illegal file name; ignored

VM.TMP appears as an object-file name. Rename the file and rerun the linker.

L4054 *filename* **: cannot find file**

The linker could not find the specified file. Enter a new filename, a new path specification, or both.

Linker Limits

The table below summarizes the limits imposed by the linker. However, your program may be adjusted so that the linker will set aside its limits to accommodate it.

Item	Limit
Symbol table	256KB
Load-time relocations	Default is 32KB. If **/EXEPACK** is used, the maximum is 512KB.
Public symbols	The range 7700 through 8700 can be used as a guideline for the maximum number of public symbols allowed; the actual maximum depends on the program.
External symbols per module	1023
Groups	Maximum number is 21, but the linker always defines DGROUP so the effective maximum is 20.
Overlays	63
Segments	128 by default; however, this maximum can be set as high as 3072 by using the **/SEGMENTS** option of the LINK command.
Libraries	32
Group definitions per module	21
Segments per module	255
Stack	64KB

Chapter 9. Converting File Formats

This chapter describes how to use EXE2BIN.EXE on your utilities diskette to convert an .EXE file to a .COM file format.

The EXE2BIN.EXE Utility

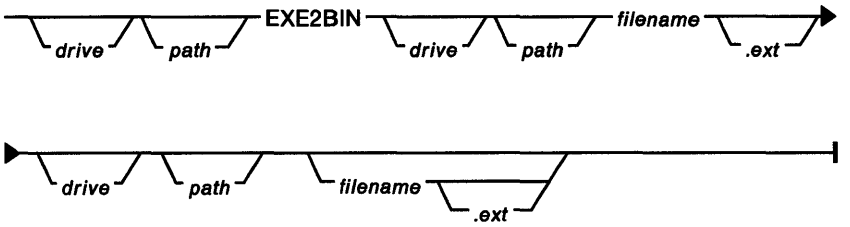
EXE2BIN is useful if you have a simple .EXE file that doesn't need the DOS preparation that an .EXE file usually requires, such as telling the program its location in memory and setting up a stack for it to use. Converting an .EXE file to a .COM format makes the file more compact and enables it to load faster.

To qualify for EXE2BIN conversion, your .EXE file has to meet these criteria:

- The file format must be a valid .EXE one, as produced by the IBM Linker.
- Actual code and data in the file (the resident portion of the program) must be less than 64KB.
- Only one segment can be declared; therefore, the file cannot define a STACK segment.
- If the file is to be loaded by the DOS 4.00 command processor, the IBM Macro Assembler ORG statement must set the location pointer of the file at 100H.
- The IBM Macro Assembler END statement must specify the first location as the starting address.

Entering Input to EXE2BIN

Input to EXE2BIN must be entered at the command line and has the following format:



Note that the only parameter required by EXE2BIN is the filename of the input file. If you specify only a filename, EXE2BIN looks for a file with an .EXE extension in the current directory of the default drive. The output file created by a successful conversion has the same filename with a .BIN extension.

For example, suppose you have a file called MYFILE.EXE in your current working directory that meets EXE2BIN's conversion criteria. To have EXE2BIN convert the file, you can enter:

```
EXE2BIN MYFILE
```

EXE2BIN creates a memory image of MYFILE.EXE and copies it to the output file. The new file named MYFILE.BIN is placed in your current working directory.

Of course, you can specify any of these optional parameters. If you specify a drive and path for your input file and do not specify an output specification, the output file is placed on the current drive and directory.

Note: This is a change from previous versions of DOS.

Two Types of Conversion

Two types of .EXE file conversion are possible, depending on how the entry point of your program is defined.

Device Drivers

The first type of conversion is for a program that will be loaded at the absolute memory address specified by a user application or by DOS 4.00 at system initialization. An example of this kind of program is a device driver.

CS:IP is not specified in the program (the .EXE file header contains 0:0 as the program's entry point). Segment fixups are allowed in this type of conversion. If segment fixups are necessary, (that is, your program contains instructions requiring segment relocation), you are prompted for the fixup value. Here are some examples of code that will cause prompting for fixup values:

- You have used the SEG operator:

```
symbol1  db  "e:\filename",0
symbol2  dw  SEG symbol1  ;explicit use of SEG
symbol3  dw  symbol1      ;implicit use of OFFSET
```

- You have used a segment name as an immediate field of instruction:

```
myseg    SEGMENT    PUBLIC

:
MOV      AX,MYSEG          ; fixup
```

The fixup value is the absolute segment at which the program is to be loaded.

The DOS 4.00 command processor is not capable of properly loading this type of program.

Standard .COM File

To produce standard .EXE files that are suitable for EXE2BIN conversion, you must use the Macro Assembler ORG statement to set the location pointer of the file at 100H and specify the first location as the start address in the END statement.

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

When EXE2BIN sees that CS:IP is specified as 0000:100H, it assumes that the file is to be run as a standard .COM file, with the location pointer set at 100H by the ORG statement. No segment fixups are possible because standard .COM files must be segment relocatable. Once the conversion is complete, you may rename the resultant file to a .COM extension. Then the command processor is capable of loading and executing the program in the same manner as the .COM programs supplied on your DOS 4.00 diskette.

If your file does not fit one of these two descriptions, or if it is similar to the standard .COM file description but has segment fixups, the following message is displayed:

```
File cannot be converted
```

This message is also displayed if the file is not a valid .EXE file.

See the IBM Macro Assembler manual for a comparison of .EXE and .COM file structures. Skeleton structures for .EXE files and .COM files are provided on the MASM diskette.

Chapter 10. Debugging a Program

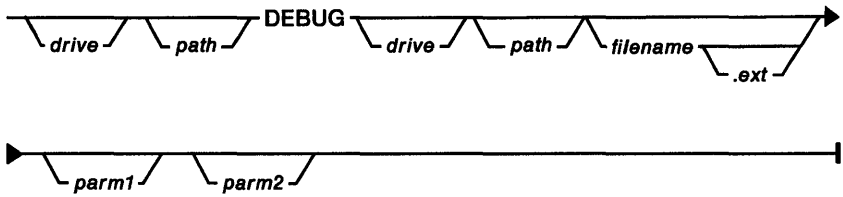
This chapter describes how to use the DEBUG.COM program on the utilities diskette to identify and fix problems in your programs.

The DEBUG Utility

DEBUG provides a controlled testing environment that enables you to monitor the execution of a program. You can make changes directly to a .COM or an .EXE file and execute the file immediately to determine whether your changes fixed a problem. You do not need to reassemble source code files first. DEBUG allows you to load, alter, or display any file and to execute object files as well.

Starting the DEBUG.COM Program

To start DEBUG, enter information in the following format:



You can enter just the DEBUG command, or you can include a file specification. The parameters *parm1* and *parm2* represent input and output specifications of the program you are debugging. For example, suppose you wanted to monitor the execution of the DOS 4.00 DISKCOMP utility. You enter:

```
DEBUG DISKCOMP.COM A: B:
```

The DEBUG program loads DISKCOMP into memory and displays the DEBUG prompt:

The hyphen (-) tells you DEBUG is ready to accept commands to alter, display, or execute the contents of the program in memory.

If you enter just DEBUG without a file specification, you can either work with the present memory contents or you can load a required file into memory using the DEBUG **Name** and **Load** commands.

Entering Commands at the DEBUG Prompt

A DEBUG command consists of a single letter, usually followed by one or more parameters. For example, the **Name** command is entered at the DEBUG prompt as a single letter followed by a file specification:

```
-N MYPROG
```

A command and its parameters can be entered in uppercase, lowercase, or a combination of both. The command and its parameters can be separated by delimiters; however, delimiters are only required between two consecutive hexadecimal values. Thus, the following **Dump** commands are equivalent:

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

A command is activated only after you press the Enter key. If you want to terminate a command and return to the DEBUG prompt, simultaneously press the Ctrl and Break keys.

For commands producing a large amount of output, you can simultaneously press the Ctrl and Num Lock keys (or Pause key if available) to suspend the display and then press any key to restart the display, or you can redirect the command's output to a file.

When DEBUG encounters a syntax error in a line, it displays the line with the error identified as follows:

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

In this example, the **Dump** command expects the second address to contain only a hexadecimal offset value. It finds the S, which is not a hexadecimal character.

DEBUG Command Summary

The table below lists the DEBUG commands and describes the debugging operations you can perform with them. Complete format descriptions and examples for each command can be found starting on page 10-6.

Command	Task Description
A (Assemble)	Assemble IBM Macro Assembler statements directly into memory.
C (Compare)	Compare the contents of two blocks of memory.
D (Dump)	Dump the contents of a portion of memory to the display or redirect it to a file.
E (Enter)	Make changes to bytes in memory.
F (Fill)	Fill a range of memory with byte values.
G (Go)	Execute the program in memory from one address to the breakpoint address and then display the next instruction.
H (Hex)	Add and subtract two hexadecimal values and display the results.
I (Input)	Display the input in the first byte next to the port.
L (Load)	Load the contents of absolute disk sectors or a file specified by the **Name** command into memory.
M (Move)	Copy the contents of a block of memory to another location.
N (Name)	Set up file control blocks and file specification information for **Load** and **Write** commands.
O (Output)	Send a byte to an output port.

Command	Task Description
P (Proceed)	Execute a subroutine call, loop instruction, interrupt, or repeat string instruction and return control to DEBUG at the next instruction.
Q (Quit)	End the DEBUG session without saving the debugged program.
R (Register)	Display the contents of registers and the settings of flags.
S (Search)	Search a range of memory for characters.
T (Trace)	Execute one or more instructions in your program and display the contents of registers and flags after each instruction.
U (Unassemble)	Translate the contents of memory into Assembler-like statements, displaying their addresses and hexadecimal values.
W (Write)	Write the debugged program to absolute disk sectors or to the original file loaded with DEBUG.
XA (Allocate)	Allocate a specified number of expanded memory pages to an EMS handle.
XD (Deallocate)	Deallocate an EMS handle.
XM (Map)	Map an EMS logical page to an EMS physical page from an EMS handle.
XS (Status)	Display the status of expanded memory.

The DEBUG Work Space

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

- The segment registers (CS, DS, ES, and SS) are set to the bottom of free memory; that is, the first segment after the end of the DEBUG program.
- The Instruction Pointer (IP) is set to hex 0100.

- The Stack Pointer (SP) is set to the end of the segment, or the bottom of the transient portion of the program loader, whichever is lower. The segment size at offset 6 is reduced by hex 100 to allow for a stack that size.
- The remaining registers (AX, BX, CX, DX, BP, SI, and DI) are set to 0. However, if you start the DEBUG program with a file specification, the CX register contains the length of the file in bytes. If the file is greater than 64KB, the length is contained in registers BX and CX (the high portion in BX).
- The initial state of the flags is:

  ```
  NV UP EI PL NZ NA PO NC
  ```

- The default disk transfer address is set to hex 80 in the code segment.

All of available memory is allocated. At this point, the loaded program is unable to allocate memory.

.EXE Files

If a file loaded by DEBUG has an extension of .EXE, DEBUG does the necessary relocation and sets the segment registers, stack pointer, and instruction pointer to the values defined in the file. The DS and ES registers, however, point to the program segment prefix at the lowest available segment. The BX and CX registers contain the size of the program that is smaller than the file size.

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

.HEX Files

If a file loaded by DEBUG has an extension of .HEX, the file is assumed to be in INTEL hex format, and is converted to executable form while being loaded.

A (Assemble) Command

Purpose
Assembles IBM Macro Assembler language statements directly into memory.

Format
A[*address*]

Parameters

address Use any of the following formats:

- A segment register plus an offset, such as CS:0100
- A segment address plus an offset, such as 4BA:0100
- An offset only, such as 100. In this case, the default segment is used.

Comments
All numeric input to the Assemble command is in hexadecimal. The assembly statements you enter are assembled into memory at successive locations, starting with the address specified in *address*. If no address is specified, the statements are assembled into the area at CS:0100, if no previous Assemble command was used, or into the location following the last instruction assembled by a previous Assemble command. After all desired statements have been entered, press the Enter key when you are prompted for the next statement to return to the DEBUG prompt.

DEBUG responds to invalid statements by displaying:

```
^error
```

and re-displaying the current assemble address.

DEBUG supports standard 8086/8088 assembly language syntax (and the 8087 instruction set), with the following rules:

- All numeric values entered are hexadecimal and can be entered as 1 through 4 characters.
- Prefix mnemonics are entered in front of the opcode to which they refer. They can also be entered on a separate line.

- The segment override mnemonics are CS:, DS:, ES:, and SS:.
- String manipulation mnemonics must specify the string size. For example, MOVSW must be used to move word strings and MOVSB must be used to move byte strings.
- The mnemonic for the far return is RETF.
- The assembler will automatically assemble short, near, or far jumps and calls depending on byte displacement to the destination address. These can be overridden with the NEAR or FAR prefix. For example:

```
0100:0500 JMP 502   ;a 2 byte short jump
0100:0502 JMP NEAR 505 ;a 3 byte near jump
0100:0505 JMP FAR 50A  ;a 5 byte far jump
```

  The NEAR prefix can be abbreviated to NE, but the FAR prefix cannot be abbreviated.

- DEBUG cannot tell whether some operands refer to a word memory location or a byte memory location. In this case, the data type must be explicitly stated with the prefix WORD PTR or BYTE PTR. DEBUG will also accept the abbreviations WO and BY. For example:

```
NEG   BYTE PTR [128]
DEC   WO [SI]
```

- DEBUG also cannot tell whether an operand refers to a memory location or to an immediate operand. DEBUG uses the common convention that operands enclosed in square brackets refer to memory. For example:

```
MOV   AX,21   ;Load AX with 21H
MOV   AX,[21]  ;Load AX with the
                contents of memory
                location 21H
```

- Two popular pseudo-instructions have also been included. The DB opcode assembles byte values directly into memory. The DW opcode assembles word values directly into memory. For example:

```
DB   1,2,3,4,"THIS IS AN EXAMPLE"
DB   'THIS IS A QUOTE: "'
DB   "THIS IS AN APOSTROPHE:'"

DW   1000,2000,3000,"BACH:"
```

A (Assemble) Command

- All forms of the register indirect commands are supported. For example:

```
ADD   BX,34[BP+2][SI-1]
POP   [BP+DI]
PUSH  [SI]
```

- All opcode synonyms are supported. For example:

```
LOOPZ   100
LOOPE   100

JA      200
JNBE    200
```

- For numeric co-processor opcodes the WAIT or FWAIT prefix must be explicitly specified. For example:

```
FWAIT FADD ST,ST(3)     ;This line will
                        ;assemble a
                        ;FWAIT prefix
FLD TBYTE PTR [BX]       ;This line will
                        ;not
```

Example

```
C>debug
-a200
08B4:0200 xor ax,ax
08B4:0202 mov [bx],ax
08B4:0204 ret
08B4:0205
```

Purpose

Compares the contents of two blocks of memory.

Format

C *range address*

Parameters

range Either of these two formats:

- An *address* followed by an offset, such as CS:100 110.
- An *address* followed by **L** *value*, where *value* is the number of hexadecimal bytes to be processed. For example, CS:100 L 10.

The limit for *range* is hexadecimal 10000 or decimal 64KB. To specify a range of 64KB within 4 hexadecimal characters, enter 0000 or 0 for *value*.

address Any of these three formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

Comments

The contents of the two blocks of memory are compared; the length of the comparison is determined from the *range*. If unequal bytes are found, their addresses and contents are displayed, in the form:

```
addr1  byte1  byte2  addr2
```

where, the first half (addr1 byte1) refers to the location and contents of the mismatching locations in *range*, and the second half (byte2 addr2) refers to the byte found in *address*.

If you enter only an offset for the beginning address of *range*, the C command assumes the segment contained in the DS register. To specify an ending address for *range*, enter it with only an offset value.

C (Compare) Command

Example

```
C  100 L20 200
```

The 32 bytes (hex 20) of memory beginning at DS:100 are compared with the 32 bytes beginning at DS:200. L20 is the range.

Purpose

Displays the contents of a portion of memory.

Format

D [*address*]

or

D [*range*]

Parameters

address Any of the following formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

range Either of these two formats:

- An *address* followed by an offset, such as CS:100 110.
- An *address* followed by **L** *value*, where *value* is the number of hexadecimal bytes to be processed. For example, CS:100 L 10.

The limit for *range* is hexadecimal 10000 or decimal 64K bytes. To specify a range of 64K bytes within 4 hexadecimal characters, enter 0000 or 0 for *value*.

Comments

The dump is displayed in two parts:

1. A hexadecimal portion. Each byte is displayed in hexadecimal.
2. An ASCII portion. The bytes are displayed as ASCII characters. Unprintable characters (ASCII 0 to 31 and 127 to 255) are indicated by a period.

D (Dump) Command

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

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

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

The Dump command has two format options.

Option 1

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

```
D address

  or

D
```

The contents are dumped starting with the specified address.

If you do not specify an address, the D command assumes the starting address is the location following the last location displayed by a previous D command. Thus, it is possible to dump consecutive 40-byte or 80-byte areas by entering consecutive D commands without parameters.

If no previous D command was entered, the location is offset hex 100 into the segment originally initialized in the segment registers by DEBUG.

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

Option 2

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

```
D range
```

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

For example:

```
D cs:100 10C
```

A 40-column display format might look like this:

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

04BA:0108  20 42 4F 52 47
                    BORG
```

E (Enter) Command

Purpose

- Replaces the contents of one or more bytes, starting at the specified address, with the values contained in the list (see Option 1).
- Displays and allows modification of bytes in a sequential manner (see Option 2).

Format

E *address* [*list*]

Parameters

address Any of the following formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

list A string of byte values. If you include a character string, enclose the characters in single or double quotation marks. To specify a quotation mark as a character within the string when it is also used to delimit the string, type it twice.

```
"These ""quotes"" are correct."
'This one''s okay, too.'
```

Comments

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

The Enter command has two format options.

Option 1

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

```
E  address list
```

For example:

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

Memory locations ds:100 through ds:104 are filled with the 5 bytes specified in the list.

Option 2

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

For example:

```
E  address
```

Enter a 1- or 2-character hexadecimal value to replace the contents of the byte; then take any one of the following actions:

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

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

2. Enter a hyphen to back up to the preceding address. A new line is displayed with the preceding address and its contents. If you want to change the contents, take option 1, above.

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

3. Press the Enter key to end the Enter command.

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

E (Enter) Command

For example:

```
E cs:100
```

might cause this display:

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

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

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

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

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

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

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

Now, suppose you want to back up and change the hex 10 to hex 6F. This is what the screen looks like after entering two hyphens and the replacement byte:

```
04BA:0100 EB.41 10.00.  BC.42-
04BA:0102 00.-
04BA:0101 10.6F_
```

Press the Enter key to end the Enter command. The hyphen prompt will appear.

Purpose

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

Format

F *range list*

Parameters

range Either of these two formats:

- An *address* followed by an offset, such as CS:100 110
- An *address* followed by **L** *value*, where *value* is the number of hexadecimal bytes to be processed. For example, CS:100 L 10.

The limit for *range* is hexadecimal 10000 or decimal 64K bytes. To specify a range of 64K bytes within 4 hexadecimal characters, enter 0000 or 0 for *value*.

list A string of byte values. If you include a character string, enclose the characters in single or double quotation marks. To specify a quotation mark as a character within the string when it is also used to delimit the string, type it twice.

```
"These ""quotes"" are correct."
'This one''s okay, too.'
```

Comments

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

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

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

F (Fill) Command

Example

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

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

Purpose

Executes the program you are debugging.

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

Format

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

Parameters

address Any of the following formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

Comments

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

The Go command has two format options.

Option 1

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

```
G [=address]
```

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

Option 2

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

```
G [=address] address
  [address...]
```

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

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

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

Notes:

1. Once a program has reached completion (DEBUG has displayed the "Program terminated normally" message), you must reload the program before it can be executed again.
2. Make sure that the address parameters refer to locations that contain valid 8086/8088 instruction codes. If you specify an address that does not contain valid instruction in the first byte, unpredictable results occur.
3. The stack pointer must be valid and have 6 bytes available for the Go command, otherwise, unpredictable results occur.
4. If only an offset is entered for a breakpoint, the G command assumes the segment contained in the CS register.
5. Do not set breakpoints at instructions in read-only memory (ROM BIOS or ROM BASIC).

For example:

```
G 102 1EF 208
```

Be careful not to set a breakpoint between a segment override indication (such as ES; alone on a line), and the instruction that the override qualifies.

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

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

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

H (Hexarithmetic) Command

Purpose

Adds the two hexadecimal values, then subtracts the second from the first. Displays the sum and difference on one line.

Format

H *value value*

Example

```
H 0F 8
0017 0007
```

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

Purpose

Inputs and displays (in hexadecimal) 1 byte from the specified port.

Format

I *portaddress*

Parameters

portaddress A 1 – 4 character hexadecimal value specifying an 8- or 16-bit port address.

Example

```
I 2F8
6B
```

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

L (Load) Command

Purpose
Loads a file or absolute disk sectors into memory.

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

Parameters

address Any of the following formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

drive A decimal number that indicates a particular drive. For example, drive A is 0, drive B is 1, and so on.

sector 1—3 character hexadecimal values that specify the starting relative sector number and the number of sectors to be loaded or written.

Note: Relative sector numbers are obtained by counting the sectors on the disk surface. The first sector on the disk is at track 0, sector 1, head 0, and is relative sector 0. The numbering continues for each sector on that track and head, then continues with the first sector on the next head of the same track. When all sectors on all heads of the track have been counted, numbering continues with the first sector on head 0 of the next track.

Comments
The maximum number of sectors that can be loaded with a single Load command is hex 80. A sector contains 512 bytes.

Note: DEBUG displays a message if a disk read error occurs. You can retry the read operation by pressing the F3 key to redisplay the Load command. Then press the Enter key.

The Load command has two format options.

Option 1

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

```
L  address drive sector sector
```

The data is read from the specified starting relative sector (first sector) and continues until the requested number of sectors is read (second sector).

Note: If you only enter an offset for the beginning address, the L command assumes the segment contained in the CS register.

For example, to load data, you might enter:

```
L DS:100 1 0F 6D
```

The data is loaded from the diskette in drive B and placed in memory beginning at DS:100. Consecutive sectors of data are transferred, 6DH (109), starting with relative sector hex 0F (15) (the 16th sector on the diskette).

Note: Option 1 cannot be used if the drive specified is a network drive.

L (Load) Command

Option 2

When issued without parameters, or with only the address parameter, use this option to load the file whose file specification is at CS:80. For example:

```
L

  or

L address
```

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

Note: If DEBUG was started with a file specification and subsequent Name commands were used, you may need to enter a new Name command for the proper file specification before issuing the Load command.

The file is loaded into memory beginning at CS:100 (or the location specified by *address*), and is read from the drive specified in the file specification, or from the default drive, if none was specified. Note that files with extensions of .COM or .EXE are always loaded at CS:100. If you specified an address, it is ignored.

The BX and CX registers are set to the number of bytes read; however, if the file being loaded has an extension of .EXE, the BX and CX registers are set to the actual program size. The file may be loaded at the high end of memory. Refer to "The DEBUG Work Space" on page 10-4 for the conditions that are in effect when .EXE or .HEX files are loaded.

For example:

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

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

Purpose

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

Format

M *range address*

Parameters

range Either of these two formats:

- An *address* followed by an offset, such as CS:100 110.
- An *address* followed by **L** *value*, where *value* is the number of hexadecimal bytes to be processed. For example, CS:100 L 10.

The limit for *range* is hexadecimal 10000 or decimal 64KB. To specify a range of 64KB within 4 hexadecimal characters, enter 0000 or 0 for *value*.

address Any of the following formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

Comments

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

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

M (Move) Command

Notes:

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

Example

```
M CS:100 L10 500
```

The 17 bytes of data from CS:100 through CS:110 are moved to the area of memory beginning at DS:500.

Purpose

- Formats file control blocks for the first two file specifications, at CS:5C and CS:6C. (Starting DEBUG with a file specification also formats a file control block at CS:5C.)

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

- All file specifications and other parameters, including delimiters, are placed exactly as entered in a parameter save area at CS:81, with CS:80 containing the number of characters entered. Register AX is set to indicate the validity of the drive specifiers entered with the first two file specifications.

Format

N [d:][path]*filename*[*.ext*]

Comments

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

Example

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

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

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

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

myprog is overwritten. The parameter area at CS:81 contains all characters entered after the **N**, including all delimiters, and CS:80 contains the count of those characters (hex 0C).

Purpose

Sends the *byte* to the specified output port.

Format

O *portaddress byte*

Parameters

portaddress A 1 – 4 character hexadecimal value specifying an 8- or 16-bit port address.

Example

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

```
O 2F8 4F
```

P (Proceed) Command

Purpose

Causes the execution of a subroutine call, a loop instruction, an interrupt, or a repeat string instruction to stop at the next instruction.

Format

P[=address][value]

Parameters

address Any of the following formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

value A 1–4 character hexadecimal value, specifying the number of instructions to execute.

Comments

When at a subroutine call, a loop instruction, an interrupt, or a repeat string instruction, issue the Proceed command to execute the instruction (perform the entire function), and return control at the next instruction. The Proceed command has the same syntax as the Trace command. Specifying P0 is the same as specifying T0.

Example

If the following instructions are executed:

```
0100    CALL    1000
0103    JC      2000
⋮
1000    XOR     AX,AX
⋮
1XXX    RET
```

And if CS:IP was pointing to the CALL 1000 instruction, typing **P** causes the execution of the subroutine and returns control to DEBUG at the JC instruction.

Purpose
Ends the DEBUG program.

Format
Q

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

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

Example
```
-Q
A>
```

R (Register) Command

Purpose

The Register command has the following three functions:

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

Format

R [*registername*]

Parameters

registername The valid names are:

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

IP refers to the instruction pointer, and F refers to the flags register.

Comments

When the DEBUG program starts, the registers and flags are set to certain values for the program being debugged. (Refer to "The DEBUG Work Space" on page 10-4.)

Display a Single Register

To display the contents of a single register, enter the register name:

```
R AX
```

The system might respond with:

```
AX F1E4
:_
```

Now you can take one of two actions: press the Enter key to leave the contents unchanged, or change the contents of the AX register by entering a 1-4 character hexadecimal value, such as hex FFF.

```
AX F1E4
:FFF_
```

Now, pressing the Enter key changes the contents of the AX register to hex 0FFF.

Display All Registers and Flags

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

```
R
```

The system responds:

```
AX=0E00  BX=00FF  CX=0007  DX=01FF  SP=039D  BP=0000  SI=005C  DI=0000
DS=3D5B  ES=3D5B  SS=3D5B  CS=3D5B  IP=011A   NV UP EI PL NZ NA PO NC
3D5B:011A CD21          INT     21
```

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

A system with an 80-column display shows:

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

A system with a 40-column display shows:

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

R (Register) Command

Display All Flags

There are eight flags, each with two-letter codes to indicate either a set condition or a clear condition. The flags appear in displays in the order shown in the following table:

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

To display all flags, enter:

```
R F
```

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

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

Now you can either press the Enter key to leave the settings unchanged or change any or all of the settings.

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

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

The changes in this example are entered in reverse order.

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

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

```
R F
```

The response is:

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

The first, third, fifth, and seventh flags are changed as requested. The second, fourth, sixth, and eighth flags are unchanged. A single flag can be changed only once for each R F command.

S (Search) Command

Purpose

Searches the *range* for the character(s) in the *list*.

Format

S *range list*

range Either of these two formats:

- An *address* followed by an offset, such as CS:100 110.
- An *address* followed by **L** *value*, where *value* is the number of hexadecimal bytes to be processed. For example, CS:100 L 10.

The limit for *range* is hexadecimal 10000 or decimal 64KB. To specify a range of 64KB within 4 hexadecimal characters, enter 0000 or 0 for *value*.

list A string of byte values. If you include a character string, enclose the characters in single or double quotation marks. To specify a quotation mark as a character within the string when it is also used to delimit the string, type it twice.

```
"These ""quotes"" are correct."
'This one''s okay, too.'
```

Comments

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

A display of the prompt without an address means that no match was found.

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

Example

If you want to search the range of addresses from CS:100 through CS:110 for hex 41, type:

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

If two matches are found the response might be:

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

If you want to search the same range of addresses for a match with the 4-byte list (41 "AB" E), enter:

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

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

T (Trace) Command

Purpose

Executes one or more instructions starting with the instruction at CS:IP, or at =*address*, if it is specified. The = must be entered. One instruction is assumed, but you can specify more than one with *value*. This command displays the contents of all registers and flags *after each* instruction executes. For a description of the display format, refer to the Register command.

Format

T [=*address*][*value*]

Parameters

address Any of the following formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

value A 1–4 character hexadecimal value, specifying the number of instructions to execute.

Comments

The display caused by the Trace command continues until *value* instructions are executed. Therefore, when tracing multiple instructions, remember you can suspend the scrolling at any time by pressing the Ctrl and the NumLock keys together, or the Pause key. Resume scrolling by entering any other character.

Notes:

1. The Trace command disables all hardware interrupts before executing the user instruction, and then re-enables the interrupts when the trap interrupt occurs following the execution of the instruction.
2. TRACE should not be used with any steps that change the contents of the 8259 interrupt mask (ports 20 and 21).

3. If you trace an INT3 instruction, the breakpoint is set at the INT3 location.

Example

```
T
```

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

```
AX=0E00  BX=00FF  CX=0007  DX=01FF  SP=039D  BP=0000  SI=005C  DI=0000
DS=3D5B  ES=3D5B  SS=3D5B  CS=3D5B  IP=011C   NV UP EI PL NZ NA PO NC
3D5B:011C CD21          INT     21
```

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

```
T 10
```

Sixteen instructions are executed (starting at CS:IP). The contents of all registers and flags are displayed after each instruction. The display stops after the 16th instruction has been executed. Displays may scroll off the screen unless you suspend the display by simultaneously pressing the Ctrl and NumLock keys, or the Pause key.

U (Unassemble) Command

Purpose

Unassembles instructions (that is, translates the contents of memory into assembler-like statements) and displays their addresses and hexadecimal values, together with assembler-like statements. For example, a display might look like this:

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

Format

U [*address*]

or

U [*range*]

Parameters

address Any of the following formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

range Either of these two formats:

- An *address* followed by an offset, such as CS:100 110.
- An *address* followed by **L** *value*, where *value* is the number of hexadecimal bytes to be processed. For example, CS:100 L 10.

The limit for *range* is hexadecimal 10000 or decimal 64KB. To specify a range of 64KB within 4 hexadecimal characters, enter 0000 or 0 for *value*.

Comments

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

Notes:

1. In all cases, the number of bytes unassembled and displayed may be slightly more than either the amount requested or the default amount. This happens because the length of the instructions vary. Therefore, unassembling the last instruction may result in more bytes than expected.
2. Make sure that the address parameters refer to locations containing valid 8086/8088 instruction codes. If you specify an address that does not contain the first byte of a valid instruction, the display will be incorrect.
3. If you enter only an offset for the starting address, the U command assumes the segment contained in the CS register.

The Unassemble command has the following two format options:

Option 1

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

```
U

  or

U address
```

Sixteen bytes are unassembled with a 40-column display. Thirty-two bytes are unassembled in 80-column mode.

Instructions are unassembled beginning with the specified address.

If you do not specify an address, the U command assumes the starting address is the location following the last instruction unassembled by a previous U command. Thus, it is possible to unassemble consecutive locations, producing continuous unassembled displays, by entering consecutive U commands without parameters.

U (Unassemble) Command

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

Option 2

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

```
U range
```

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

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

For example:

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

The display response may be:

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

The same display appears if you enter:

```
U 04BA:100 L 7

     or

U 04BA:100 L 8

     or

U 04BA:100 L 9
```

Purpose

Writes the data being debugged to disk.

Format

W [*address* [*drive sector sector*]]

Parameters

address Any of the following formats:

- A segment register plus an offset, such as CS:0100.
- A segment address plus an offset, such as 4BA:0100.
- An offset only, such as 100. In this case, the default segment is used.

drive A decimal number that indicates a particular drive. For example, drive A is 0, drive B is 1, and so on.

sector 1–3 character hexadecimal values that specify the starting relative sector number and the number of sectors to be loaded or written.

Note: Relative sector numbers are obtained by counting the sectors on the disk surface. The first sector on the disk is at track 0, sector 1, head 0, and is relative sector 0. The numbering continues for each sector on that track and head, then continues with the first sector on the next head of the same track. When all sectors on all heads of the track have been counted, numbering continues with the first sector on head 0 of the next track.

Comments

No more than hex 80 sectors can be written with a single Write command. A sector contains 512 bytes.

DEBUG displays a message if a disk write error occurs. You can retry the write operation by pressing the F3 key to re-display the Write command, then press the Enter key.

W (Write) Command

The Write command has two format options.

Option 1

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

```
W address drive sector sector
```

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

Notes:

1. Be extremely careful when you write data to absolute sectors because an erroneous sector specification destroys whatever was on the disk at that location.
2. If only an offset is entered for the beginning address, the W command assumes the segment is contained in the CS register.
3. Remember, the starting sector and the sector count are both specified in *hexadecimal*.
4. Option 1 cannot be used if the specified drive is a network drive.

For example:

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

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

Option 2

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

```
W

  or

W address
```

When issued as shown above, the Write command writes the file (whose file specification is at CS:80) to disk.

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

Note: If DEBUG was started with a file specification and subsequent Name commands were used, you may need to enter a new Name command for the proper file specification before issuing the Write command.

In addition, the BX and CX registers must be set to the number of bytes to be written. They may have been set properly by the DEBUG or Load commands, but were changed by a Go or Trace command. You must be certain the BX and CX registers contain the correct values.

The file beginning at CS:100, or at the location specified by *address*, is written to the diskette in the drive included in the file specification or the default drive if no drive was specified.

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

Note: An error message is issued if you try to write a file with an extension of .EXE or .HEX. These files are written in a specific format that DEBUG cannot support.

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

1. RENAME the file to an extension other than .EXE or .HEX.
2. Load the file into memory using the DEBUG or Load command.
3. Modify the file as needed in memory, but do not try to execute it with the Go or Trace commands. Unpredictable results would occur.
4. Write the file back using the Write command.
5. RENAME the file to its correct name.

XA (EMS Allocate) Command

Purpose
Allocates a specified number of expanded memory pages to a handle.

Format
XA *count*

Comments
The *count* indicates the number of 16K pages to allocate. If the amount of expanded memory identified by *count* is available, a message is displayed, indicating that a handle has been created. The XS (EMS Status) command can be used to display the number of expanded memory pages that are available.

Example
To allocate two EMS pages, enter:

```
XA 2
```

If two pages of memory are available, a message like this is displayed:

```
Handle created = 000E
```

Purpose
Deallocates a handle.

Format
XD *handle*

Comments
The *handle* identifies the number of the handle to be deallocated. If the number is valid, a message is displayed, indicating the handle has been deallocated. The XS (EMS Status) command can be used to display the handles currently being used.

Example
To deallocate a handle when you have only one allocated, you can enter:

```
XD 000E
```

If the handle deallocation is successful, you receive a message like this:

```
Handle 000E deallocated.
```

XM (EMS Map) Command

Purpose

Maps an EMS logical page to an EMS physical page from an EMS handle.

Format

XM */page ppage handle*

Comments

The */page* specifies the number of the handle's logical page that is to be mapped. The *ppage* is the number of the physical page to be mapped to. The *handle* is the EMS allocated label used to reference a group of logical pages. If syntax items are valid, a message is displayed indicating that the logical page has been mapped to the physical page.

Example

To map a logical page to a physical page using handle 0001, you can enter:

```
XM 1 0 1
```

If the mapping is successful, you receive this message:

```
Logical page 01 mapped to physical page 00.
```

Purpose

Displays the status of expanded memory.

Format

XS

Comments

The following expanded memory information is displayed:

```
Handle %1 has %2 pages allocated

⋮
Physical page %1 = Frame segment %2

⋮
%1 of a total %2 EMS pages have been allocated
%1 of a total %2 EMS handles have been allocated
```

Example

A line is displayed for each handle allocated with its associated logical page count.

DEBUG Error Messages

The following error messages are produced by the DEBUG utility:

Access denied	The result of attempting to Write (W) to a read-only file.
Disk error reading drive %1	An invalid parameter was entered on the Load (L) command or an error occurred on issuing the Load (L) command.
Disk error writing drive %1	An invalid parameter was entered on the Write (W) command or an error occurred on issuing the Write (W) command.
EMS hardware/software failure	The result of an EMS command. Tells the user EMS is not functioning properly.
EMS not installed	The result of an EMS command. Tells user EMS is not installed.
^ Error	Points to the offending operand in an error condition.
Error in EXE or HEX file	The EXE or HEX file are in error.
EXE and HEX files cannot be written	A file in EXE or HEX format cannot be written to a disk.
EXEC failure	The execution of the requested file failed.
File creation error	The result of attempting to Write (W) to a system or hidden file.
File not found	Issued from the Load (L) command when a file is not found for loading.

Free pages exceeded	The result of an EMS command. Tells the user that the request has exceeded the amount of free EMS pages available.
Handle not found	The result of an EMS command. Tells the user an EMS handle was not found.
Incorrect DOS version	Incorrect version of DEBUG for the DOS version running.
Insufficient memory	Not enough memory to Load (L) the specified file.
Insufficient space on disk	Out of disk space for a Write (W) command.
Invalid drive specification	The drive referenced by the Name (N) and Load (L) command is invalid; that is, it does not exist.
Logical Page out of range	The result of an EMS command. Tells the user that the logical page requested is not in the range of possible pages.
Missing or invalid EMS parameter	The result of an EMS command. Tells the user a missing or invalid parameter was entered.
No free handles	The result of an EMS command. Tells the user that there are no more EMS handles available for allocation.
Parameter error	The result of an EMS command. Tells the user that an EMS parameter is in error.

Physical Page out of range	The result of an EMS command. Tells the user that the physical page requested is not in the range of possible pages.
Program terminated normally	An Interrupt 20H has been encountered, signalling program termination.
Total pages exceeded	The result of an EMS command. Tells the user that the total EMS pages have been exceeded.
Write (W) error, no destination defined	Attempt to Write (W) to a file that has not yet been Named (N).
Write protect error writing drive %1	A Write (W) to a write-protected disk caused an error.
Writing %1 bytes	Reports the number of bytes written to a file when the Write (W) command is issued.

Chapter 11. Writing an Installable Device Driver

This chapter provides guide and system architecture information to support successful creation of an installable device driver.

We recommend you use the information provided in this chapter in conjunction with the information provided in the VDISK.ASM file on the utilities diskette. This file contains the fully documented source code for the VDISK device driver, which emulates a fixed disk in RAM.

Note: The example on the utilities diskette does not reflect the current level of VDISK.SYS..

Types of Device Drivers

A device driver is a memory image file or an .EXE file that contains the code needed to implement a device. DOS 4.00 allows two types of device drivers to be installed:

- Character device drivers
- Block device drivers.

Character Device Drivers

Character device drivers perform character I/O in a serial manner and have names such as CON, AUX, CLOCK$. You can open handles or FCBs to perform input and output to character devices. Because character device drivers have only one name, they support only one device.

Block Device Drivers

Block device drivers are the fixed disk or diskette drives on the system. They perform random I/O in pieces called blocks, which are usually the physical sector size of the disk. Block devices are not named as character devices are and you cannot open them. Instead they are mapped using the drive letters such as A, B, and C.

A single block device driver can be responsible for one or more disk or diskette drives. For example, the first block device driver in the device chain may define four units such as A, B, C, and D. The second device driver may define three units: E, F, and G. The limit is 26 devices with the letters A through Z assigned to drives. The position of the driver in the chain determines the way in which the drive units and drive letters correspond.

Support for Media Greater than 32MB

A block device driver that runs in the DOS 4.00 environment can be written to process 32-bit sector numbers. This ability provides support for fixed disk media greater than 32MB. This support is described in these sections:

- "The Device Driver Header" on page 11-4
- "Build BPB Request" on page 11-16
- "Input and Output Requests" on page 11-20.

DOS 4.00 also allows users to install their own device drivers that support media greater than 32MB.

How DOS 4.00 Installs Device Drivers

DOS 4.00 installs device drivers dynamically at startup time by reading and processing the DEVICE command in CONFIG.SYS. For example, if you have written a device driver called DRIVER1, include the following command in CONFIG.SYS:

```
device=driver1
```

The DOS 4.00 device interface links device drivers together in a chain, permitting you to add device drivers for optional devices.

Each device must be initialized. The device driver's initialization interrupt routine is called once when the device is installed. The initialization routine returns the location in memory of the ending address of the device driver. After setting the ending address field, a character device driver sets the status word and returns.

DOS 4.00 processes installed character device drivers before handling default devices. To have DOS 4.00 install a new CON device (for example, in the device driver header's Name/Unit field) name the

device CON and set the standard input device and standard output device bits in the attribute field. Because DOS 4.00 installs drivers anywhere in memory, care must be taken in any references to locations not in the segment. Your driver will not always be loaded at the same memory location each time.

Block devices are installed in the same manner as character devices, above. Block devices return additional information such as the number of units. This number identifies the devices' logical names. For example, if the current maximum logical device letter is F at the time of the install call, and the block device driver initialization routine returns three logical units, the logical names of the devices are G, H, and I. The mapping is determined by the position of the driver in the device list and the number of units associated with the device. The number of units returned by INIT overrides the value in the name/unit field of the device header.

A block device also returns a pointer to a BIOS parameter block (BPB) array. This is a pointer to an array of *n* word pointers, where *n* is the number of units defined. If all the units are the same, the array is able to point to the same BIOS parameter block, thus saving space. The array must be protected below the ending address pointer set by the return.

The BPB contains information pertinent to the devices such as the sector size and the number of sectors per allocation unit. The sector size in the BPB cannot be greater than the maximum allowed size set by DOS 4.00 initialization.

A block device returns the media descriptor byte passed to devices to report which parameters DOS 4.00 is using for a particular drive unit.

The Basic Parts of a Device Driver

A device driver is a memory image file or an .EXE file containing the code needed to implement a device. All DOS 4.00 installable device drivers have these things in common:

- A device driver header, which identifies the device to DOS 4.00 and defines the strategy and interrupt entry points. Since a device driver is simply loaded and does not use a program

segment prefix, the device header must be located at physical location 0 of the device driver (ORG 0 or no ORG statement).

- A strategy routine, which saves a pointer to the Request Header.
- The interrupt routines, which perform the requested operation.

The Device Driver Header

A device driver requires a device header containing the following:

Field	Length
Pointer to next device header	DWORD
Attribute	WORD
Pointer to device strategy routine	WORD
Pointer to device interrupt routine	WORD
Name/unit field	8 BYTES

Pointer to Next Device Header

The device driver header is a pointer to the device header of the next device driver. It is a doubleword field set by DOS 4.00 at the time the device driver is loaded. The first word is an offset and the second word is the segment.

If you are loading only one device driver, set the device header field to -1 before loading the device. If you are loading more than one device driver, set the first word of the device header field to the offset of the next device driver's header. Set the device header field of the last device driver to -1.

Attribute Field

The attribute field identifies the device to DOS 4.00.

Bit 15
Bit 15 identifies whether the device is a block device or a character device. If bit 15 is set to 0, this indicates a block device. Setting bit 15 to 1 indicates a character device. Note how the setting of bit 15 affects the interpretation of the setting of the bits below.

Bit 14
Bit 14, for both character and block devices, tells DOS 4.00 whether the device driver can handle control strings through IOCtl 44H, AL=2 through AL=5. Set bit 14 to 1 if control strings can be processed. IOCtl subfunctions permit the device driver to interpret the information passed to it, such as setting a baud rate or changing form lengths, without performing standard reads and writes. Set bit 14 to 0 if control strings cannot be processed. DOS 4.00 will return an error if an IOCtl is issued to send or receive control strings and bit 14 is set to 0.

Bit 13
Bit 13 is used for both block and character devices. For block devices, set bit 13 to 0 if the media is an IBM format. Set bit 13 to 1 if the media is a non-IBM format. For character devices, set bit 13 to 0 if the driver supports output-until-busy. Set bit 13 to 1 if it does not.

With the support of output-until-busy, the device driver will send characters to the device if the device is ready. If the device is not ready, the device driver will immediately return an error.

Bit 12
Bit 12 is reserved.

Bit 11
Set bit 11 if the device driver can handle removable media. This bit is called the open/close removable media bit.

Bits 10 through 7
Bits 10 through 7 are reserved.

Bit 6
Bit 6 is the generic IOCtl bit for both character and block device drivers. If this bit is set to 1, the device driver supports generic IOCtl function calls. Setting this bit to 1 also indicates support of the Get/Set Logical Drive function for a block device driver.

Bits 5 and 4
Bits 5 and 4 are reserved.

Bit 3
Set bit 3 to 1 if the character device is a clock device; set bit 3 to 0 if it is not.

Bit 2
Set bit 2 to 1 if the character device is the NUL device; set bit 2 to 0 if it is not. Setting the bit tells DOS 4.00 whether the NUL device is being used. The NUL device cannot be reassigned.

Bit 1
If bit 15 is set to 0 for a character device, set bit 1 to 1 to indicate that the character device is the current standard output device. If bit 15 is set to 1 for a block device, set bit 1 to 1 to indicate support for 32-bit sector numbers; otherwise 16-bit sector number support is assumed.

Bit 0
Set bit 0 to 1 if the character device is the current standard input device; set bit 0 to 0 if it is not the current standard input device.

Pointers to Strategy Routine and Interrupt Routines

When DOS 4.00 passes a request to a device driver, it calls the device driver twice. These two fields point to the first and second entry points: the strategy routine and the interrupt routine. The fields are word values, so they must be in the same segment as the device header.

Name/Unit Field

These 8-byte fields identify a character device by name or a block device by unit. A character device name is left-justified followed by spaces, if necessary. For block devices, although DOS 4.00 automatically fills in this field with the value of number of units returned by INIT call, you may choose to place the number of units in the first place.

The Strategy Routine

DOS 4.00 calls a device driver at the strategy routine at first, passing in a request packet the information describing what DOS 4.00 wants the device driver to do.

The strategy routine does not perform the request but queues the request or saves a pointer to the request packet.

The Interrupt Routines

DOS 4.00 calls the device driver's interrupt routine with no parameters immediately after the strategy routine returns. An interrupt routine's function is to perform the operation based on the queued request, process any data in the request packet, and set up information being returned to DOS 4.00.

It is the responsibility of the device driver to preserve the system state. For example, the device driver must save all registers on entry and restore them on exit. The stack maintained by DOS 4.00 is used to save all registers. If more stack space is needed, it is the device driver's responsibility to allocate and maintain an additional stack.

All calls to device drivers are FAR calls. FAR returns should be executed to return to DOS 4.00.

How DOS 4.00 Passes a Request

DOS 4.00 passes a pointer in ES:BX to the request packet. The packet consists of a request header that contains information common to all requests, followed by data pertinent to the request being made.

The structure of the request header is shown below.

Field	Length
Length of the request header and subsequent data	BYTE
Unit code for block devices only	BYTE
Command code	BYTE
Status	WORD
Reserved	8-BYTE
Data	VARIABLE

Length Field

The length field identifies the length of the request header and subsequent data in bytes.

Unit Code Field

The unit code field identifies the requesting unit in a block device driver. If a block device driver has three units defined, for example, the possible values for the unit code field are 0, 1, or 2.

Command Code Field

The command code identifies the request. See "Responding to Requests" on page 11-9 for a list of command code values and request descriptions.

Status Field

The status word field is zero on entry and is set by the driver interrupt routine on return.

Bit 15
Bit 15 is the error bit. If bit 15 is set to 1, the low order 8 bits of the status word (7-0) indicate the error code.

Bits 14 – 10
Bits 14 through 10 are reserved.

Bit 9
Bit 9 is the busy bit. As a response to status request call, character device drivers can set the busy bit to indicate whether or not a device is ready to perform input and output requests. Block device drivers can set the busy bit to indicate removable or nonremovable media. See "Character Input and Output Status Requests" on page 11-23 and "Removable Media Request" on page 11-25 for more information about the calls.

Bit 8
Bit 8 is the done bit. If set, the operation is complete. The driver sets the done bit to 1 when it exits.

Bits 7 – 0

Bits 7 through 0 are the low order 8 bits of the status word. If bit 15 is set, bits 7 through 0 contain the error codes. The error codes and errors are:

Codes	Meaning
00	Write protect violation
01	Unknown unit
02	Device not ready
03	Unknown command
04	CRC error
05	Bad drive request structure length
06	Seek error
07	Unknown media
08	Sector not found
09	Printer out of paper
0A	Write fault
0B	Read fault
0C	General failure
0D	Reserved
0E	Reserved
0F	Invalid disk change

Responding to Requests

Each request packet that is passed to the device driver contains a command code value in the request header to tell the driver which function to perform. The table below contains the DOS 4.00 device interface command code values and the functions to be performed when these values are passed with data. Note that some of these functions are specific to either a block device or a character device.

Following this table are detailed descriptions of request data structures and what the interrupt routines are expected to do. Some of these descriptions pertain to more than one command code.

Command Code	Request Description	Device Type
0	Initialization	Both
1	Media check	Block
2	Build BPB	Block
3	IOCtl input (called only if bit 14 of attribute is set to 1)	Both
4	Input (read)	Both
5	Nondestructive input no wait	Character
6	Input status	Character
7	Input flush	Character
8	Output (write)	Both
9	Output (write with verify)	Block
10	Output status	Character
11	Output flush	Character
12	IOCtl output (called only if bit 14 of attribute is set to 1	Both
13	Device open (called only if bit 11 of attribute is set to 1)	Both
14	Device close (called only if bit 11 of attribute is set to 1)	Both
15	Removable media (called only if bit 11 of attribute is set to 1)	Block
19	Generic IOCtl Request (called only if bit 6 of attribute is set to 1)	Both
23	Get logical device (called only if bit 6 of attribute is set to 1)	Block
24	Set logical device (called only if bit 6 of attribute is set to 1)	Block

Initialization Request

Command Code = 0

Field	Length
Request header	13 – BYTE
Number of units (not set by character devices)	BYTE
Ending address of resident program code	DWORD
Pointer to BPB array (not set by character devices) pointer to remainder of arguments	DWORD
Drive number	BYTE
CONFIG.SYS Error Message control flag	WORD

The driver must do the following:

- Set the number of units (block devices only).
- Set up the pointer to the BPB array (block devices only).
- Perform any initialization code (to modems, printers, and others).
- Set the ending address of the resident program code.
- Set the status word in the request header.

To obtain information passed from CONFIG.SYS to a device driver at initialization time, the BPB pointer field points to a buffer containing the information passed in CONFIG.SYS following the =. This string may end with either a carriage return (0DH) or a linefeed (0AH). This information is read-only. Only system calls 01H – 0CH and 30H can be issued by the initialization code of the driver.

The last byte parameter contains the drive letter for the first unit of a block driver. For example, 0 = A, 1 = B and so forth.

If an initialization routine determines that it cannot set up the device and wants to terminate without using any memory, use the following procedure:

- Set the number of units to 0.
- Set the ending address offset to 0.
- Set the ending address segment to the code segment (CS).

If there are multiple device drivers in a single memory image file, the ending address returned by the last initialization called is the one DOS 4.00 uses. IBM recommends that all device drivers in a single memory image file return the same ending address.

If initialization of your device driver fails, and you want the system to display the **Error in Config.Sys line #** error message, set the CONFIG.SYS error message control flag to a non-zero value.

Media Check Request

Command code = 1

Field	Length
Request header	13 – BYTE
Media descriptor from DOS 4.00	BYTE
Return	BYTE
A pointer to the previous volume ID (if bit 11 = 1 and disk change is returned)	DWORD

When the command code field is 1, DOS 4.00 calls Media Check for a drive unit and passes its current Media Descriptor byte. See "Media Descriptor Byte" on page 11-14. Media Check returns one of the following:

- Media not changed
- Media changed
- Not sure
- Error code.

The driver must perform the following:

- Set the status word in the request header.
- Set the return byte to:

  -1 for "media changed"
  0 for "not sure"
  1 for "media not changed."

The driver uses the following method to determine how to set the return byte:

- If the media is nonremovable (a fixed disk), set the return byte to 1.
- If less than 2 seconds since last successful access, set the return byte to 1.
- If changeline not available, set the return byte to 0.
- If changeline is available but not active, set the return byte to 1.
- If the media byte in the new BPB does not match the old media byte, set the return byte to -1.

- If the current volume ID matches the previous volume ID, or if the serial number matches the previous serial number, set the return byte to 0.

Media Descriptor Byte

Currently the media descriptor byte has been defined for some media types. This byte should be identical to the media byte if the device has the non-IBM format bit off. These predefined values are:

Media descriptor

byte —>	1	1	1	1	1	x	x	x
bits —>	7	6	5	4	3	2	1	0

Bit	Meaning	
0	1 = 2-sided	0 = not 2-sided
1	1 = 8 sector	0 = not 8 sector
2	1 = removable	0 = not removable
3—7	must be set to 1	

Note: An exception to the above is that the media descriptor byte value of F0, which is used to indicate any media types not defined, and F9, which is used for 5.25-inch media with 2 sides and 15 sectors/tracks.

Examples of current DOS 4.00 media descriptor bytes:

Disk Type	# Sides	Sectors/ Track	Media Descriptor
Fixed disk	--	--	F8H
5.25 inch	2	15	F9H
5.25 inch	1	9	FCH
5.25 inch	2	9	FDH
5.25 inch	1	8	FEH

Disk Type	# Sides	Sectors/ Track	Media Descriptor
5.25 inch	2	8	FFH
8 inch	1	26	FEH
8 inch	2	26	FDH
8 inch	2	8	FEH
3.5 inch	2	9	F9H
3.5 inch	2	18	F0H

To determine whether you are using a single-sided or a double-sided diskette, attempt to read the second side. If an error occurs, you may assume the diskette is single-sided. Media descriptor F0H may be used for those media types not described earlier. Programs should not use the media descriptor values to distinguish media. DOS 4.00 internal routines use information in the BIOS parameter block (BPB) to determine the media type of IBM-formatted diskettes. These media descriptor bytes do not necessarily indicate a unique media type.

For 8-inch diskettes:

- FEH (IBM 3740 Format) — Single-sided, single-density, 128 bytes per sector, soft-sectored, 4 sectors per allocation unit, 1 reserved sector, 2 FATs, 68 directory entries, 77*26 sectors.
- FDH (IBM 3740 Format) — Double-sided, single-density, 128 bytes per sector, soft-sectored, 4 sectors per allocation unit, 4 reserved sectors, 2 FATs, 68 directory entries, 77*26*2 sectors.
- FEH — Double-sided, double-density, 1024 bytes per sector, soft sectored, 1 sector per allocation unit, 1 reserved sector, 2 FATs, 192 directory entries, 77*8*2 sectors.

Build BPB Request
Command Code = 2

Field	Length
Request header	13-BYTE
Media descriptor from DOS 4.00	BYTE
Transfer address (buffer address)	DWORD
Pointer to BPB table	DWORD

DOS 4.00 calls Build BPB (BIOS Parameter Block) under the following two conditions:

- If "Media Changed" is returned
- If "Not Sure" is returned, there are no used buffers. Used buffers are buffers with changed data not yet written to the disk.

The driver must do the following:

- Set the pointer to the BPB
- Set the status word in the request header.

The device driver must determine the media type that is in the unit to return the pointer to the BPB table. In previous versions of IBMBIO, the FAT ID byte determined the structure and layout of the media. The FAT ID byte has only eight possible values (F8 through FF), so, as new media types are invented, the available values will soon be exhausted. With the varying media layouts, DOS 4.00 needs to be aware of the location of the FATs and directories before it asks to read them.

The following paragraphs explain the method DOS 4.00 uses to determine the media type.

The information relating to the BPB for a particular media is kept in the boot sector for the media. The format of the boot sector is as follows:

Field	Length
A 2-byte short JMP instruction (EBH), followed by a NOP instruction (90H).	WORD
Product name and version	8 BYTES
Bytes per sector; must be power of 2	WORD
Sectors per allocation unit; must be power of 2	BYTE
Reserved sectors starting at logical sector 0	WORD
Number of FATs	BYTE
Maximum number of root directory entries	WORD
Total number of sectors in media including the boot sector, FAT areas, and directories	WORD
Media descriptor	BYTE
Number of sectors occupied by a FAT	WORD
Sectors per track	WORD
Number of heads	WORD
Number of hidden sectors	WORD

The BPB information contained in the boot sector starts with the Bytes per Sector entry. The last three words are intended to help the device driver identify the media. The number of heads is useful for supporting different multihead drives with the same storage capacity but a different number of surfaces. The number of hidden sectors is useful for supporting drive partitioning schemes.

For drivers that support volume identification and disk change, this call causes a new volume identification to be read from the disk. This call indicates that the disk has changed in a permissible manner.

To handle the partition that is bigger than 32MB, or one that starts beyond or crosses the 32MB boundary, DOS 4.00 defines an extended BPB structure. Depending on the size of the media, you can use either the existing BPB or the extended one, which contains an additional DWORD field to indicate the size of the partition in sectors.

Bit 1 of the attribute field in the block device driver header indicates whether the device can process 32-bit sector numbers. Set bit 1 to indicate 32-bit support.

Field	Length
Bytes per sector	WORD
Sectors per allocation unit	BYTE
Reserved sectors starting at logical sector 0	WORD
Number of FATs — 0 if not a FAT system	BYTE
Maximum number of root directory entries	WORD
Total number of sectors in the media. This field is used to define a partition that is less than 32MB. Setting this field to 0 indicates to use the total (32-bit) number of sectors in the media below.	WORD
Media descriptor	BYTE
Number of sectors occupied by a FAT	WORD
Sectors per track	WORD
Number of heads	WORD
Number of hidden sectors	DWORD
Total (32-bit) number of sectors in the media. This field is used to define a partition that is greater than 32MB, or one that crosses the 32MB boundary.	DWORD

The extended BPB is a superset of the traditional BPB structure. To achieve the maximum compatibility between this structure and that of the traditional BPB, DOS 4.00 uses the following rule:

- If (the number of hidden sectors plus the total number of sectors in the media) is greater than 64KB, use the 32-bit total number of sectors in the media entry (DWORD).
- Otherwise, use the **Total number of sectors in the media** entry (WORD).

A boot record exists at the beginning of each disk partition and each extended DOS 4.00 partition volume. DOS 4.00 automatically creates the extended boot record. The format of the extended boot record is:

Field	Length
A 2-byte short JMP instruction (EBH) followed by a NOP instruction (90H).	WORD
Product name and version	8 BYTES
Extended BPB	25 BYTES
Physical drive number	BYTE
Reserved	BYTE
Extended boot record signature	BYTE
Volume serial number	DWORD
Volume label	11 BYTES
Reserved	8 BYTES

Note: The value of Extended boot record signature is 29H. The value of the physical drive number is always 0H or 80H.

On all requests to extended drivers with a sector number in their request headers, the sector number is a DWORD. The standard DOS 4.00 block device drivers set the attribute bit 1 for 32-bit support.

For each call to a device driver, DOS 4.00 checks to see if the starting sector number passed in the request can be supported by the device driver. If this value is greater than 64K for an old-style device driver, DOS 4.00 returns an **unknown media** (07H) device driver error.

Input and Output Requests
Command Codes = 3, 4, 8, 9, 12

Field	Length
Request header	13 – BYTE
Media descriptor byte	BYTE
Transfer address (buffer address)	DWORD
Byte/sector count	WORD
Starting sector number (If -1, use DWORD starting sector number. This entry has no meaning for a character device.)	WORD
For DOS 3.0 to DOS 4.00, pointer to the volume identification if error code 0FH is returned	DWORD
Starting 32-bit sector number. (Use this entry to the block device driver with the attribute bit 1 set.)	DWORD

The DOS 4.00 **Input/Output** request structure can process 32-bit sector numbers, providing support for media of more than 4 billion sectors.

The driver must do the following:

- Set the status word in the request header
- Perform the requested function
- Set the actual number of sectors or bytes transferred.

No error checking is performed on an IOCtl call. However, the driver must set the return sector or byte count to the actual number of bytes transferred.

Under certain circumstances the block device driver may be asked to do a WRITE operation of 64KB that seems to be a *wraparound* of the transfer address in the device driver request packet. This occurs because an optimization is added to the WRITE code in DOS 4.00. It will only happen on WRITEs that are within a sector size of 64KB on files that are being extended past the current end of file. The block device driver is allowed to ignore the balance of the WRITE that wraps around. For example, a WRITE of 10000H bytes of sectors with a transfer address of XXXX:1 ignores the last two bytes.

Remember that a program using DOS 4.00 function calls cannot request an input or output operation of more than FFFFH bytes because a wrap around in the transfer buffer segment must occur. You can ignore bytes that would have wrapped around in the transfer segment.

If the driver returns an error code of 0FH (Invalid Disk Change), it must provide a DWORD pointer to an ASCIIZ string identifying the correct volume ID and the system prompts the user to reinsert the disk.

The reference count of open files on the disk maintained by OPEN and CLOSE calls allows the driver to determine when to return error 0FH. If there are no open files (reference count=0) and the disk has been changed, the I/O is valid, and error 0FH is not returned. If there are open files (reference count > 0) and the disk has been changed, an error 0FH situation may exist. DOS 4.00 IBMDOS.COM will request an OPEN or CLOSE function only if SHARE is loaded.

Nondestructive Input No Wait Request
Command Code = 5

Field	Length
Request header	13-BYTE
Byte read from device	BYTE

The driver must do the following:

- Return a byte from the device.
- Set the status word in the request header.

If the character device returns busy bit = 0, meaning there are characters in buffer, the next character that would be read is returned. This character is not removed from the input buffer, (that is, nondestructive input). This call allows DOS 4.00 to look ahead one input character.

Character Input and Output Status Requests

Command Codes = 6, 10

Field	Length
Request header	13-BYTE

The driver must do the following:

- Perform the requested function.
- Set the busy bit.
- Set the status word in the request header.

The busy bit is set differently for output and input.

Output on Character Devices

If the busy bit is 1 on return, a write request would wait for completion of a current request. If the busy bit is 0, no request is waiting or running. Therefore, a write request would start immediately.

Input on Character Devices with a Buffer

If the busy bit is 1 on return, a read request goes to the physical device. If the busy bit is 0, characters are in the device buffer and a read returns quickly. This also indicates that the user has typed something. DOS 4.00 assumes that all character devices have a type-ahead input buffer. Devices that do not have this buffer should always return busy = 0 so that DOS 4.00 does not loop endlessly, waiting for information to be put in a buffer that does not exist.

Character Input and Output Flush Requests

Command Codes = 7, 11

Field	Length
Request header	13-BYTE

This call tells the driver to flush (terminate) all pending requests of which it has knowledge. Its primary use is to flush the input queue on character devices.

The driver must set the status word in the Request Header upon return.

Open and Close Requests

Command Codes = 13, 14

Field	Length
Request header	13-BYTE

These calls are designed to give the device information about current file activity on the device if bit 11 of the attribute word is set.

On block devices, these calls can be used to manage local buffering. The device can keep a reference count. Every OPEN increases the reference count. Every CLOSE decreases the device reference count. When the reference count is 0, there are no open files on the device. Therefore, the device should flush buffers inside the device to which it has written because the user can change the media on a removable media drive. If the media has been changed, reset the reference count to 0 without flushing the buffers.

These calls are more useful on character devices. The OPEN call can send a device an initialization string. On a printer, the call can send a string to set the font or the page size so the printer is always in a known state at the start of an I/O stream.

Similarly, the CLOSE call can send a post string, such as a form feed, at the end of an I/O stream.

Using IOCtl to set the preliminary and ending strings provides a flexible mechanism for serial I/O device stream control.

Removable Media Request

Command Code = 15

Field	Length
Request header	13-BYTE

To use this call, set bit 11 of the attribute field to 1. Block devices can use this call only by way of a subfunction of the IOCtl function call (44H).

This call is useful because it notifies a utility if it is dealing with a removable or nonremovable media drive. For example, the FORMAT utility needs to know whether a drive is removable or nonremovable because it displays different versions of some prompts.

The information is returned in the busy bit of the status word. If the busy bit is 1, the media is nonremovable. If the busy bit is 0, the media is removable.

No error bit checking is performed. It is assumed that this call always succeeds.

Generic IOCTL Request

Command Code = 19

Field	Length
Request header	13-BYTE
Major function	BYTE
Minor function	BYTE
Contents of SI	WORD
Contents of DI	WORD
Pointer to Generic IOCTL request packet	DWORD

The driver must:

- Support the functions described under Generic IOCtl request
- Maintain its own track table (TrackLayout).

See Appendix C, "I/O Control for Devices (IOCtl)" on page C-1 for a description of the functions provided by generic IOCtl requests.

Get Logical Device Request

Command Code = 23

Field	Length
Request header	13-BYTE
Input (unit code)	BYTE
Command code	BYTE
Status	WORD
Reserved	DWORD

Set Logical Device Request

Command Code = 24

Field	Length
Request header	13-BYTE
Input (unit code)	BYTE
Command code	BYTE
Status	WORD
Reserved	DWORD

CLOCK$ Device Driver Example

This feature is a "Real-Time Clock" Board. To allow this board to be integrated into the system for TIME and DATE, a special device driver is specified by an attribute word, the CLOCK$ device. This device driver defines and performs functions like any other character device driver. Most functions will be set done bit, reset error bit, return. When a read or write to this device occurs, 6 bytes are transferred. The first 2 bytes are a word, which is the count of days since 1-1-80. The third byte is minutes; the fourth is hours; the fifth is hundredths of seconds; and the sixth is seconds. Reading the CLOCK$ device gets the date and time; writing to it sets the date and time.

Part 3. Appendixes

Appendix A. DOS 4.00 Interrupts

This chapter contains information to support use of the DOS 4.00 interrupts.

DOS 4.00 reserves interrupt types 20H to 3FH for its use. Absolute memory locations 80H to FFH are reserved by DOS 4.00. All interrupt values are in hexadecimal.

20H Program Terminate

Issue interrupt 20H to exit from a program. This vector transfers to the logic in DOS 4.00 to restore the terminate address, the Ctrl-Break address, and the critical error exit address to the values they had on entry to the program. All file buffers are flushed and all handles are closed. You should close all files changed in length (see function call 10H and 3EH) before issuing this interrupt. If the changed file is not closed, its length, date, and time are not recorded correctly in the directory.

For a program to pass a completion code or an error code when terminating, it must use either function call 4CH (Terminate a Process) or 31H (Terminate Process and Stay Resident). These two methods are preferred over using interrupt 20H, and the codes returned by them can be interrogated in batch processing. See function call 4CH for information on the ERRORLEVEL subcommand of batch processing.

Important: Before you issue interrupt 20H, your program must ensure that the CS register contains the segment address of its program segment prefix.

21H Function Request

Refer to each function call issued within 21H in Appendix B, "DOS 4.00 Function Calls" on page B-1.

22H Terminate Address

Control transfers to the address at this interrupt location when the program terminates. This address is copied into the program's Program Segment Prefix at the time the segment is created. You should not issue this interrupt; the EXEC function call does this for you.

23H Ctrl-Break Exit Address

If the user presses the Ctrl and Break keys during standard input, standard output, standard printer, or asynchronous communications adapter operations, an interrupt 23H is executed. If BREAK is on, the interrupt 23H is checked on most function calls (except calls 06H and 07H). If the user-written Ctrl-Break routine saves all registers, it may end with a return-from-interrupt instruction (IRET) to continue program execution.

If the user-written interrupt program returns with a long return, the carry flag is used to determine whether the program will be ended. If the carry flag is set, the program is ended, otherwise execution continues (as with a return by IRET).

If the user-written Ctrl-Break interrupt uses function calls 09H or 0AH, then ^C, carriage-return and linefeed are output. If execution is continued with an IRET, I/O continues from the start of the line.

When the interrupt occurs, all registers are set to the value they had when the original function call to DOS 4.00 was made. There are no restrictions on what the Ctrl-Break handler is allowed to do, including DOS 4.00 function calls, as long as the registers are unchanged if IRET is used.

When the program creates a new segment and loads in a second program it changes the Ctrl-Break address. The termination of the second program and return to the first causes the Ctrl-Break address to be restored to the value it had before execution of the second program. It is restored from the second program's Program Segment Prefix. You should not issue this interrupt.

24H Critical Error Handler Vector

A critical error is returned when a DOS function cannot be performed. This error is frequently caused by a hardware condition, such as the printer being out of paper, a diskette drive door open, or a diskette out of space. When a critical error occurs within DOS 4.00, control is transferred with an interrupt 24H. On entry to the error handler, AH will have its bit 7 = 0 (high-order bit) if the error was a disk error (the most common occurrence), bit 7 = 1 if it was not.

BP:SI contains the address of a Device Header Control Block from which additional information can be retrieved. *See page* A-7.

The registers are set up for a retry operation, and an error code is in the lower half of the DI register with the upper half undefined. The error codes follow:

Error Code	Meaning
0	Attempt to write on write-protected diskette
1	Unknown unit
2	Drive not ready
3	Unknown command
4	Data error (CRC)
5	Bad request structure length
6	Seek error
7	Unknown media type
8	Sector not found
9	Printer out of paper
A	Write fault
B	Read fault
C	General failure

The user stack is in effect and contains the following from top to bottom:

IP	DOS 4.00 registers from issuing INT 24H
CS	
FLAGS	
AX	User registers at time of original
BX	INT 21H request
CX	
DX	
SI	

DI
BP
DS
ES
IP From the original interrupt 21H
CS from the user to DOS 4.00
FLAGS

The registers are set such that if an IRET is executed, DOS 4.00 responds according to (AL) as follows:

(AL) = 0 ignore the error.
= 1 retry the operation.
= 2 terminate the program through interrupt 22H.
= 3 fail the system call in progress.

Note: Be careful when choosing ignore as a response because this causes DOS 4.00 to believe that an operation has completed successfully when it may not have.

To return control from the critical error handler to a user error routine, the following should occur:

Before an INT 24H occurs:

1. The user application initialization code should save the INT 24H vector and replace the vector with one pointing to the user error routine.

When the INT 24H occurs:

2. When the user error routine receives control, it should push the flag register onto the stack, and then execute a CALL FAR to the original INT 24H vector saved in step 1.
3. DOS 4.00 gives the appropriate prompt, and waits for the user input (Abort, Retry, Fail or Ignore). After the user input, DOS 4.00 returns control to the user error routine at the instruction following the CALL FAR.
4. The user error routine can now do any tasks necessary. To return to the original application at the point the error occurred, the error routine needs to execute an IRET instruction. Otherwise, the user error routine should remove the IP, CS, and Flag

registers from the stack. Control can then be passed to the desired point.

Disk Errors

If it is a hard error on disk (AH bit 7=0), register AL contains the failing drive number (0 = drive A, and so on). AH bits 0−2 indicate the affected disk area and whether it was a read or write operation, as follows:

Bit 0=0 if read operation,
1 if write operation
Bits 2-1 (affected disk area)
0 0 DOS 4.00 area
0 1 file allocation table
1 0 directory
1 1 data area

AH bits 3−5 indicate which responses are valid. They are:

Bit 3=0 if FAIL is not allowed
=1 if FAIL is allowed
Bit 4=0 if RETRY is not allowed
=1 if RETRY is allowed
Bit 5=0 if IGNORE is not allowed
=1 if IGNORE is allowed

Handling of Invalid Responses

If IGNORE is specified (AL=0) and IGNORE is not allowed (bit 5=0), make the response FAIL (AL=3).

If RETRY is specified (AL=1) and RETRY is not allowed (bit 4=0), make the response FAIL (AL=3).

If FAIL is specified (AL=3) and FAIL is not allowed (bit 3=0), make the response END (AL=2).

Other Errors

If AH bit 7 = 1, the error occurred on a character device, or was the result of a bad memory image of the FAT. The device header passed in BP:SI can be examined to determine which case exists. If the attribute byte high-order bit indicates a block device, then the error was a bad FAT. Otherwise, the error is on a character device.

If a character device is involved, the contents of AL are unpredictable and the error code is in DI as above.

Notes:

1. Retry five times before giving this routine control for disk errors. When the errors are in the FAT or a directory, retry three times.
2. For disk errors, this exit is taken only for errors occurring during an interrupt 21H function call. It is not used for errors during an interrupt 25H or 26H.
3. This routine is entered in a disabled state.
4. All registers must be preserved.
5. This interrupt handler should refrain from using DOS 4.00 function calls. If necessary, it may use calls 01H through 0CH, 30H, and 59H. Use of any other call destroys the DOS 4.00 stack and leaves DOS 4.00 in an unpredictable state.
6. The interrupt handler must not change the contents of the device header.
7. If the interrupt handler handles errors itself rather than returning to DOS 4.00, it should restore the application program's registers from the stack, remove all but the last three words on the stack, then issue an IRET. This will return to the program immediately after the INT 21H that experienced the error. Note that if this is done, DOS 4.00 will be in an unstable state until a function call higher than 0CH is issued; therefore, it is not recommended.

   The recommended way to end a critical error is to use FAIL and then test the extended error code of the INT 21H.
8. IGNORE requests (AL = 0) are converted to FAIL for critical errors that occur on FAT or DIR sectors.

9. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for information on how to obtain additional error information.
10. For DOS 4.00, IGNORE requests (AL=0) are converted to FAIL requests for certain critical errors (50−79).

The device header pointed to by BP:SI is formatted as follows:

DWORD Pointer to next device (FFFFH if last device)
WORD Attributes: Bit 15 = 1 if character device = 0 if block device If bit 15 is 1: Bit 0 = 1 if current standard input Bit 1 = 1 if current standard output Bit 2 = 1 if current NULL device Bit 3 = 1 if current CLOCK device Bit 14 is the IOCtl bit
WORD pointer to device driver strategy entry point.
WORD pointer to device driver interrupt entry point.
8-BYTE character device named field for block devices. The first byte is the number of units.

To tell if the error occurred on a block or character device, look at bit 15 in the attribute field (WORD at BP:SI+4).

If the name of the character device is desired, look at the eight bytes starting at BP:SI+10.

25H/26H Absolute Disk Read/Write

Interrupt vectors 25H and 26H transfer control to the device driver. They have been extended to allow direct access to media greater than 32MB in size. Their use of the CX register is what distinguishes them from the conventional 25H and 26H interrupts. Note that if the conventional format parameters are used in an attempt to access media greater than 32MB, an error code of 0207H is returned in AX.

The request for extended 25H or 26H is:

```
        MOV     AL,DRIVE        ; Drive number to process
                                ; 0 = A
                                ; 1 = B
                                ; 2 = C . . .
        LDS     BX,PACKET       ; Parameter list
        MOV     CX,-1           ; Indicates extended format
        INT     25H or 26H      ; Issue request to DOS 4.00
        POP     AX              ; Discard stack word
        JC      ERROR           ; Error code returned in AX

PACKET  LABEL   BYTE            ; Control packet
        DD      RBA             ; RBA of first sector
                                ; (0 origin)
        DW      COUNT           ; Number of sectors to I/O
        DD      BUFFER          ; Data buffer
```

On return, the original flags are still on the stack (put there by the INT instruction). This is necessary because return information is passed back in the current flags. Be sure to pop the stack to prevent uncontrolled growth.

Warning: If disk I/O handled by this interrupt exceeds the limit imposed by the 64KB direct memory access boundary, unpredictable results can occur. We recommend you carefully check the sector size and the number of sectors to be read or written to before issuing this call.

The number of sectors specified is transferred between the given drive and the transfer address. *Logical sector numbers* (LSN) are obtained by numbering each sector sequentially starting from track 0, head 0, sector 1 (logical sector 0) and continuing along the same head, then to the next head until the last sector on the last head of the track is counted. Thus, logical sector 1 is track 0, head 0, sector 2; logical sector 2 is track 0, head 0, sector 3; and so on. Numbering continues with sector 1 on head 0 of the next track. Note that although the sectors are sequentially numbered (for example, sectors 2 and 3 on track 0 in the example above), they may not be physically adjacent on the disk, because of interleaving. Note that the mapping is different from that used by DOS version 1.10 for dual-sided diskettes.

All registers except the segment registers are destroyed by this call. If the transfer was successful, the carry flag (CF) is 0. If the transfer was not successful CF = 1 and (AX) indicate the error as follows. (AL) is the DOS 4.00 error code that is the same as the error code returned

in the low byte of DI when an interrupt 24H is issued, and (AH) contains:

80H	Attachment failed to respond
40H	SEEK operation failed
08H	Bad CRC on diskette read
04H	Requested sector not found
03H	Write attempt on write-protected diskette
02H	Error other than types listed above

Warning: Before issuing this interrupt to removable media, the media in the drive must be established correctly. This can be accomplished by issuing either an INT 21H Generic IOCTL (AH=44H), with a request to return the BPB that BUILD BPB returns; or an INT 21H Get Current Directory (AH=47H).

27H Terminate but Stay Resident

This vector is used by programs that are to remain resident when COMMAND.COM regains control.

DOS 4.00 function call 31H is the preferred method to cause a program to remain resident, because this allows return information to be passed. It allows a program larger than 64KB to remain resident. After initializing itself, the program must set DX to its last address plus one, relative to the program's initial DS or ES value (the offset at which other programs can be loaded), and then execute an interrupt 27H. DOS 4.00 then considers the program as an extension of itself, so the program is not overlaid when other programs are executed. This concept is useful for loading programs such as user-written interrupt handlers that must remain resident.

Notes:

1. This interrupt must *not* be used by .EXE programs that are loaded into the high end of memory.
2. This interrupt restores the interrupt 22H, 23H, and 24H vectors in the same manner as interrupt 20H. Therefore, it cannot be used to install permanently resident Ctrl-Break or critical error handler routines.

3. The maximum size of memory that can be made resident by this method is 64KB.
4. Memory can be used more efficiently if the block containing a copy of the environment is deallocated before terminating. This can be done by loading ES with the segment contained in 2C of the PSP, and issuing function call 49H (Free Allocated Memory).
5. DOS 4.00 function call 4CH allows the terminating program to pass a completion (or error) code to DOS 4.00, which can be interpreted within batch processing (see function call 31H).
6. Terminate-but-stay-resident function calls do not automatically close files.

28H – 2EH Reserved for DOS 4.00

These interrupts are reserved for DOS 4.00 use.

2FH Multiplex Interrupt

Interrupt 2FH is the multiplex interrupt. A general interface is defined between two processes. The specific application using interrupt 2FH defines specific functions and parameters.

Every multiplex interrupt handler is assigned a specific multiplex number. The multiplex number is specified in the AH register. The specific function that the handler is to perform is specified in the AL register. Other parameters are placed in the other registers, as needed. The handlers are chained into the interrupt 2FH interrupt vector. There is no defined method for assigning a multiplex number to a handler. You must pick one. To avoid a conflict if two applications choose the same multiplex number, the multiplex numbers used by an application should be patchable.

The multiplex numbers AH = 00H through BFH are reserved for DOS 4.00. Applications should use multiplex numbers C0H through FFH.

Note: When in the chain for interrupt 2FH, if your code calls DOS 4.00 or if you execute with interrupts enabled, your code must be reentrant and recursive.

Function Codes

AH = 1

AH = 1 is the resident part of PRINT.

The following table contains the function codes that you can specify in AL to request the resident portion of print to perform a specific function:

Function Codes	Description
0	Get installed state
1	Submit file
2	Cancel file
3	Cancel all files
4	Status
5	End of status

Print Error Codes

The following table contains the error codes that are returned from the resident portion of print.

Error Codes	Description
1	Invalid function
2	File not found
3	Path not found
4	Too many open files
5	Access denied

Error Codes	Description
6	Queue full
9	Busy
12	Name too long
15	Invalid drive

AL=0 Get Installed State

This call must be defined by all interrupt 2FH handlers. It is used by the caller of the handler to determine if the handler is present. On entry, AL=0, AH=1. On return, AL contains the installed state as follows:

AL=0 Not installed, permissible to install

AL=1 Not installed, not permissible to install

AL=FF Installed

AL=1 Submit File

On entry, AL=1, AH=1, and DS:DX points to the submit packet. A submit packet contains the level (BYTE) and a pointer to the ASCIIZ string (DWORD in offset segment form). The level value for DOS 4.00 is 0. The ASCIIZ string must contain the drive, path, and filename of the file you want to print. The filename cannot contain global filename characters. *AL=2 Cancel File*

On entry, AL=2, AH=1, and DS:DX points to the ASCIIZ string for the print file you want to cancel. Global filename characters are allowed in the filename. *AL=3 Cancel all Files*

On entry, AL=3 and AH=1.

AL=4 Status

This call holds the jobs in the print queue so that you can scan the queue. Issuing any other code releases the jobs. On entry, AL=4, AH=1. On return, DX contains the error count. The error count is the number of consecutive failures PRINT had while trying to output the last character. If there are no failures, the number is 0. DS:SI points to the print queue. The print queue consists of a series of filename entries. Each entry is 64 bytes long. The first entry in the queue is the file currently being printed. The end of the queue is marked by a queue entry having a null as the first character.

AL=5 End of Status: Issue this call to release the queue from call 4. On entry, AL=5 and AH=1. On return, AX contains the error codes. For information on the error codes returned, refer to "Print Error Codes" on page A-11.

AL=F8–FF Reserved by DOS 4.00

AH=6

AH=6 is the resident part of ASSIGN.
The Get Installed State function
(AL=0) is supported.

AH=8H, 10H, 12H, 13H Reserved by DOS 4.00

AH=B7H

AH=B7H is the resident part of APPEND.
The Get Installed State function
(AL=0) is supported.

AL=2 Get APPEND version
This call is for distinguishing between
the PC LAN APPEND and the DOS 4.00 APPEND.
On return, if AX=FFFFH then the DOS 4.00
APPEND is loaded.

AL=4 Get APPEND Path Pointer
(DOS 4.00 APPEND only)
On return ES:DI points to the
currently active APPEND path.

AL = 6 Get APPEND Function State
(DOS 4.00 APPEND only)

BX is returned with bits set indicating
if APPEND is currently enabled and what
functions are in effect.

Bit	Function in effect if bit is on
0	APPEND enabled
13	/PATH
14	/E
15	/X

Note: The functions in effect do not change whether or not APPEND is disabled.

AL = 7 Set function state (DOS 4.00 APPEND
only)

On input BX is the new setting for all functions.
The suggested procedure is to get the current function
state, turn on or turn off the desired bits, then
use this call to set the function state.

AL = 11H Set Return Found Name State (DOS 4.00 APPEND
only)

On request AL = 17, a process system state flag is set. If this flag is set, then on the next ASCIIZ 3DH, 43H or 6CH function call within Interrupt 21H that APPEND processes, APPEND returns the name it finds to the application filename buffer. This name may be different from the one the application offered. The application must provide enough space for the found name. After APPEND has processed an

Interrupt 21H, it resets the Return Found Name state. An example of this process follows:

```
        MOV     AH,0B7H         ; Indicate APPEND
        MOV     AL,0            ; Get installed state
        INT     2FH
        CMP     AL,0            ; APPEND installed?
        JE      NOT_INSTALLED

        MOV     AH,0B7H         ; Indicate APPEND
        MOV     AL,2            ; Get APPEND version
        INT     2FH
        CMP     AX,-1           ; DOS version?
                PC_LAN_APPEND   ; AX<> -1 means PC LAN
        JNE
APPEND

; The following functions are valid only if DOS 4.00 APPEND

        MOV     AH,0B7H         ; Indicate APPEND
        MOV     AL,4            ; Get APPEND path pointer
        INT     2FH

                                ; ES:DI = address of APPEND path
                                ; (Buffer is 128 characters long)

        MOV     AH,0B7H         ; Indicate APPEND
        MOV     AL,6            ; Get function state
        INT     2FH
                                ; BX = function state
                                ; 8000H = /X is on
                                ; 4000H = /E is on
                                ; 2000H = /PATH is on
                                ; 0001H = APPEND enabled
                                ; If off, similar to null
                                ; APPEND path
                                ; Set on by any non-status
                                ; occurrence of APPEND

        MOV     AH,0B7H         ; Indicate APPEND
        MOV     AL,7            ; Set function state
        MOV     BX,state        ; New state
        INT     2FH

        MOV     AH,0B7H         ; Indicate APPEND
        MOV     AL,11H          ; Set Return Found
                                ; Name state
        INT     2FH
```

Example 2FH Handler

```
MYNUM           DB      X ; X = The specific AH
                          ; multiplex number.
INT_2F_NEXT     DD      ? ; Chain location
INT_2F:

ASSUME DS:NOTHING,ES:NOTHING,SS:NOTHING

  CMP     AH,MYNUM
  JE      MINE
  JMP     INT_2F_NEXT   ; Chain to next
                        ; 2FH Handler

MINE:

  CMP     AL,0F8H
  JB      DO_FUNC
  IRET                  ; IRET on reserved
                        ; functions

DO_FUNC:

  OR      AL,AL
  JNE     NON_INSTALL   ; Non Get Installed
                        ; State request
  MOV     AL,0FFH       ; Say I'm here
  IRET                  ; All done

NON_INSTALL:
:
```

Installing the Handler

The following example contains the functions necessary to install a handler:

```
   MOV     AH,MYNUM
   XOR     AL,AL
   INT     2FH                     ; Ask if already
                                   ; installed
   OR      AL,AL
   JZ      OK_INSTALL

BAD_INSTALL:                       ; Handler already  installed .sp 3
OK_INSTALL:                        ; Install my
                                   ; handler

   MOV     AL,2FH
   MOV     AH,GET_INTERRUPT VECTOR
   INT     21H                      ; Get multiplex
                                    ; vector
   MOV     WORD PTR INT_2F NEXT+2,ES
   MOV     WORD PTR INT_2F_NEXT,BX
   MOV     DX,OFFSET INT_2F
   MOV     AL,2FH
   MOV     AH,SET_INTERRUPT_VECTOR
   INT     21H                      ; Set multiplex
                                    ; vector
⋮
```

30H-3FH Reserved for DOS 4.00

These interrupts are reserved for DOS 4.00 use.

Appendix B. DOS 4.00 Function Calls

Number	Function Name
00H	Program terminate
01H	Console input with echo
02H	Display output
03H	Auxiliary input
04H	Auxiliary output
05H	Printer output
06H	Direct console I/O
07H	Direct console input without echo
08H	Console input without echo
09H	Display string
0AH	Buffered keyboard input
0BH	Check standard input status
0CH	Clear keyboard buffer, invoke a keyboard function
0DH	Disk reset
0EH	Select disk
0FH	Open file
10H	Close file
11H	Search for first entry
12H	Search for next entry
13H	Delete file
14H	Sequential read
15H	Sequential write
16H	Create file
17H	Rename file
18H	Reserved by DOS 4.00
19H	Current disk
1AH	Set disk transfer address
1BH	Allocation table information
1CH	Allocation table information for specific device
1DH	Reserved by DOS 4.00
1EH	Reserved by DOS 4.00
1FH	Reserved by DOS 4.00
20H	Reserved by DOS 4.00
21H	Random read
22H	Random write
23H	File size
24H	Set relative record field
25H	Set interrupt vector

26H	Create new program segment
27H	Random block read
28H	Random block write
29H	Parse filename
2AH	Get date
2BH	Set date
2CH	Get time
2DH	Set time
2EH	Set/reset verify switch
2FH	Get disk transfer address
30H	Get DOS 4.00 version number
31H	Terminate process and remain resident
32H	Reserved by DOS 4.00
33H	Get/Set system value
34H	Reserved by DOS 4.00
35H	Get interrupt vector
36H	Get disk free space
37H	Reserved by DOS 4.00
38H	Get/set country dependent information
39H	Create subdirectory (MKDIR)
3AH	Remove subdirectory (RMDIR)
3BH	Change current directory (CHDIR)
3CH	Create a file (CREAT)
3DH	Open a file
3EH	Close a file handle
3FH	Read from a file or device
40H	Write to a file or device
41H	Delete a file from a specified directory (UNLINK)
42H	Move file read/write pointer (LSEEK)
43H	Change file mode (CHMOD)
44H	I/O control for devices (IOCtl)
45H	Duplicate a file handle (DUP)
46H	Force a duplicate of a file handle (FORCDUP)
47H	Get current directory
48H	Allocate memory
49H	Free allocated memory
4AH	Modify allocated memory blocks (SETBLOCK)
4BH	Load or execute a program (EXEC)
4CH	Terminate a process (EXIT)
4DH	Get return code of a subprocess (WAIT)
4EH	Find first matching file (FIND FIRST)
4FH	Find next matching file (FIND NEXT)
50H	Reserved by DOS 4.00

51H	Reserved by DOS 4.00
52H	Reserved by DOS 4.00
53H	Reserved by DOS 4.00
54H	Get verify setting
55H	Reserved by DOS 4.00
56H	Rename a file
57H	Get/set a file's date and time
58H	Reserved for DOS 4.00
59H	Get extended error
5AH	Create unique file
5BH	Create new file
5CH	Lock/unlock file access
5DH	Reserved by DOS 4.00
5E00H	Get machine name
5E02H	Set printer setup
5E03H	Get printer setup
5F02H	Get redirection list entry
5F03H	Redirect device
5F04H	Cancel redirection
60H	Reserved by DOS 4.00
61H	Reserved by DOS 4.00
62H	Get PSP address
63H	Reserved by DOS 4.00
64H	Reserved by DOS 4.00
65H	Get extended country information
66H	Get/set global code page
67H	Set handle count
68H	Commit file
69H	Reserved by DOS 4.00
6AH	Reserved
6BH	Reserved by DOS 4.00
6CH	Extended open/create

Using DOS 4.00 Function Calls

Most function calls require input to be passed to them in registers. After setting the appropriate register values, issue the function calls in either of the following ways:

- The preferred method is to place the function number in AH and issue interrupt 21H.

- Place the function number in AH and execute a call to offset 50H in your program segment prefix.

Program Code Fragments

In each of the function call descriptions in this chapter, the input, output and method of use are described using a small program code fragment. These fragments are written in IBM PC Assembler Language.

.COM Programs

The descriptions assume that the program is an .EXE, not a .COM program. If a .COM program is desired, do not include either of the following instructions:

```
  MOV ES,SEG -
or
  MOV DS,SEG -
```

Notes:

1. Some FCB function calls do not permit invalid characters (0DH – 29H).
2. Device names cannot end in a colon.
3. The contents of the AX register can be altered by any of the function calls. Even though no error code is returned in AX, it is possible that AX has been changed.

DOS 4.00 Registers

DOS 4.00 uses the following registers, pointers, and flags when executing interrupts and function calls:

Register Definition	General Registers
AX	Accumulator (16-bit)
AH	Accumulator high-order byte (8-bit)
AL	Accumulator low-order byte (8-bit)

Register Definition	General Registers
BX	Base (16-bit)
BH	Base high-order byte (8-bit)
BL	Base low-order byte (8-bit)
CX	Count (16-bit)
CH	Count high-order byte (8-bit)
CL	Count low-order byte (8-bit)
DX	Data (16-bit)
DH	Data high-order (8-bit)
DL	Data low-order (8-bit)
Flags	OF,DF,IF,TF,SF,ZF,AF,PF,CF

Register Definition	Pointers
SP	Stack pointer (16-bit)
BP	Base pointer (16-bit)
IP	Instruction pointer (16-bit)

Register Definition	Segment Registers
CS	Code segment (16-bit)
DS	Data segment (16-bit)
SS	Stack segment (16-bit)
ES	Extra segment (16-bit)

Register Definition	Index Registers
DI	Destination index (16-bit)
SI	Source index (16-bit)

Register Numbering Convention

Each register is 16 bits long and is divided into a high and low byte. Each byte is 8 bits long. The bits are numbered from right to left. The low byte contains bits 0 through 7 and the high byte contains bits 8 through 15. The chart below shows the hexadecimal values assigned to each bit.

	High Byte	Low Byte
Bit	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Hex value	8 4 2 1 8 4 2 1	8 4 2 1 8 4 2 1

DOS 4.00 Internal Stack

When DOS 4.00 gains control, it switches to an internal stack. User registers are preserved unless information is passed back to the requester as indicated in the specific requests. The user stack needs to be sufficient to accommodate the interrupt system. It is recommended that the user stack be 200H in addition to what the user needs.

Responding to Errors

Handle function calls report an error by setting the carry flag and returning the error code in AX. FCB function calls report an error by returning FFH in AL.

Extended error support (59H) provides a common set of error codes and specific error information such as error classification, location, and recommended action. In most critical cases, applications can analyze the error code and take specific action. Recommended actions are intended for programs that do not understand the error

codes. Programs can take advantage of extended error support both from interrupt 24H critical error handlers and after issuing interrupt 21H function calls. Do not code to specific error codes.

Extended Error Codes

Hexadecimal Code	Decimal Code	Meaning
01H	1	Invalid function number
02H	2	File not found
03H	3	Path not found
04H	4	Too many open files (no handles left)
05H	5	Access denied
06H	6	Invalid handle
07H	7	Memory control blocks destroyed
08H	8	Insufficient memory
09H	9	Invalid memory block address
0AH	10	Invalid environment
0BH	11	Invalid format
0CH	12	Invalid access code
0DH	13	Invalid data
0EH	14	Reserved
0FH	15	Invalid drive was specified
10H	16	Attempt to remove the current directory
11H	17	Not same device
12H	18	No more files
13H	19	Attempt to write on write-protected diskette
14H	20	Unknown unit
15H	21	Drive not ready
16H	22	Unknown command
17H	23	Cyclic redundancy check (CRC) — part of diskette is bad
18H	24	Bad request structure length
19H	25	Seek error
1AH	26	Unknown media type
1BH	27	Sector not found
1CH	28	Printer out of paper

Hexadecimal Code	Decimal Code	Meaning
1DH	29	Write fault
1EH	30	Read fault
1FH	31	General failure
20H	32	Sharing violation
21H	33	Lock violation
22H	34	Invalid disk change
23H	35	FCB unavailable
24H	36	Sharing buffer overflow
25H	37	Reserved by DOS 4.00
26H	38	Unable to complete file operation
27H–31H	39–49	Reserved by DOS 4.00
50H	80	File exists
51H	81	Reserved
52H	82	Cannot make directory entry
53H	83	Fail on INT 24
54H	84	Too many redirections
55H	85	Duplicate redirection
56H	86	Invalid password
57H	87	Invalid parameter
58H	88	Network data fault
59H	89	Function not supported by network
5AH	90	Required system component not installed

Extended Error Codes

Hexadecimal Code	Decimal Code	Meaning
32H	50	Network request not supported
33H	51	Remote computer not listening
34H	52	Duplicate name on network
35H	53	Network path not found
36H	54	Network busy
37H	55	Network device no longer exists

Hexadecimal Code	Decimal Code	Meaning
38H	56	NETBIOS command limit exceeded
39H	57	System error; NETBIOS error
3AH	58	Incorrect response from network
3BH	59	Unexpected network error
3CH	60	Incompatible remote adapter
3DH	61	Print queue full
3EH	62	Not enough space for print file
3FH	63	Print file was cancelled
40H	64	Network name was deleted
41H	65	Access denied
42H	66	Network device type incorrect
43H	67	Network name not found
44H	68	Network name limit exceeded
45H	69	NETBIOS session limit exceeded
46H	70	Sharing temporarily paused
47H	71	Network request not accepted
48H	72	Print or disk redirection is paused
49H – 4FH	73 – 79	Reserved

Error Classes

Hexadecimal Value	Decimal Value	Description
01H	1	**Out of Resource:** Example: out of space or channels.
02H	2	**Temporary Situation:** Something expected to disappear with time. This is not an error condition, but a temporary situation such as a locked file.
03H	3	**Authorization:** Permission problem.
04H	4	**Internal:** Internal error in system software. Probable problem with system software rather than a user or system failure.
05H	5	**Hardware Failure:** A serious problem not the fault of user program.

Hexadecimal Value	Decimal Value	Description
06H	6	**System Failure:** Serious failure of system software not the fault of the user, such as missing or incorrect configuration files.
07H	7	**Application Program Error:** Inconsistent requests.
08H	8	**Not Found:** File or item not found. Inconsistent requests.
09H	9	**Bad Format:** File or value in invalid format or type; unsuitable.
0AH	10	**Locked:** Locked file or item.
0BH	11	**Media:** Media failure such as incorrect disk, CRC error, or defective media.
0CH	12	**Already Exists:** Duplication error, such as declaring a machine name that already exists.
0DH	13	**Unknown:** Classification does not exist or is inappropriate.

Actions

Hexadecimal Code	Decimal Code	Description
01H	1	**Retry:** Retry a few times, then prompt user to determine if the program should continue or be terminated.
02H	2	**Delay Retry:** Retry several times after pause, then prompt user to determine if the program should continue or be terminated.
03H	3	**User:** If the input was entered by a user, advise reentry. The error, however, may have occurred in the program itself, such as a bad drive letter or bad filename specification.

Hexadecimal Code	Decimal Code	Description
04H	4	**Abort:** Abort application with cleanup. The application cannot proceed, but the system is in an orderly state and it is safe to stop the application.
05H	5	**Immediate Exit:** Stop application immediately without clearing registers. Do not use the application to close files or update indexes, but exit as soon as possible.
06H	6	**Ignore:** Ignore.
07H	7	**Retry After User Intervention:** The user needs to perform some action such as removing a diskette and inserting a different one. Then retry the operation.

Locus (Location)

Hexadecimal Value	Decimal Value	Description
01H	1	**Unknown:** Not specific; not appropriate.
02H	2	**Block Device:** Related to random access mass disk storage.
03H	3	**Net:** Related to the network.
04H	4	**Serial Device:** Related to serial devices.
05H	5	**Memory:** Related to random access memory.

00H —
Program Terminate

Purpose
Stops the execution of a program.

Example
```
        MOV     AH,00H   ; Function Call - Terminate Program
        INT     21H      ; Issue request to DOS
; No return
```

Comments
The terminate, Ctrl-Break, and critical error exit addresses are restored to the values they had on entry to the terminating program, from the values saved in the program segment prefix. All file buffers are flushed and the handles opened by the process are closed. Any files that have changed in length and not closed are not recorded properly in the directory. Control transfers to the terminate address. This call performs exactly the same function as interrupt 20H. It is the program's responsibility to ensure that the CS register contains the segment address of its program segment prefix control block before calling this function.

Function 4CH — Terminate a Process is the preferred method for ending a program.

Purpose

Waits for a character to be read at the standard input device (unless one is ready), then echoes the character to the standard output device and returns the character in AL.

Example

```
        MOV     AH,01H          ; Function Call - keyboard input
        INT     21H             ; Issue request to DOS
        MOV     Char,AL         ; Save character
        CMP     AL,0            ; Extended character ?
        JNE     Normal_Char     ; No!
        MOV     AH,01H          ; Function Call - keyboard input
        INT     21H             ; Issue request to DOS
        MOV     ExtChar,AL      ; Save extended character
Normal_Char:

  -----

  Character     LABEL   WORD    ; Complete character
  Char          DB      ?       ; Character buffer
  ExtChar       DB      ?       ; Character buffer
```

Comments

The character is checked for a Ctrl-Break. If Ctrl-Break is detected, an interrupt 23H is executed.

For function call 01H, extended ASCII codes require two function calls. The first call returns 00H as an indicator that the next call will return an extended code.

02H —
Display Output

Purpose

Outputs the character in DL to the standard output device.

Example

```
    MOV     AH,02H          ; Function Call - Display Output
    MOV     DL,Char         ; Get character to display
    INT     21H             ; Issue request to DOS

-----

CHAR        DB      ?       ; Character buffer
```

Comments

If the character in DL is a backspace (08), the cursor is moved left one position (nondestructive). If a Ctrl-Break is detected after the output, an interrupt 23H is executed.

Purpose

Waits for a character from the standard auxiliary device, then returns that character in AL.

Example

```
        MOV     AH,03H          ; Function Call - Auxiliary Input
        INT     21H             ; Issue request to DOS
        MOV     Char,AL         ; Save character

-------

CHAR    DB      ?               ; Character buffer
```

Comments

Auxiliary (AUX) support is unbuffered and noninterrupt driven.

At startup, DOS 4.00 initializes the first auxiliary port to 2400 baud, no parity, one-stop bit, and 8-bit word.

The auxiliary function calls (03H and 04H) do not return status or error codes. For greater control, you should use the ROM BIOS routine (interrupt 14H) or write an AUX device driver and use IOCtl.

04H —
Auxiliary Output

Purpose

Outputs the character in DL to the standard auxiliary device.

Example

```
        MOV     AH,04H          ; Function Call - Auxiliary Output

        MOV     DL,Char         ; Get character to output
        INT     21H             ; Issue request to DOS
                                ; Nothing returned

-------

CHAR    DB      ?               ; Character buffer
```

Comments

If the character in DL is a backspace (08), the cursor is moved left one position (nondestructive). If a Ctrl-Break is detected after the output, an interrupt 23H is executed.

Purpose

Outputs the character in DL to the standard printer device.

Example

```
        MOV     AH,05H          ; Function Call - Printer Output
        MOV     DL,Char         ; Get character to output
        INT     21H             ; Issue request to DOS
                                ; Nothing returned

--------

CHAR    DB      ?               ; Character buffer
```

06H —
Direct Console I/O

Purpose

Gets a character from the standard input device if one is ready, or outputs a character to the standard output device.

Examples

```
In_loop:
   MOV     AH,06H          ; Function Call - Direct Console I/O
   MOV     DL,-1           ; 0FFH for input
   INT     21H             ; Issue request to DOS
   JZ      In_loop         ; No character yet on input
   MOV     Char,AL         ; Save character
   CMP     AL,0            ; Extended Character ?
   JNE     Normal_Char     ; No!
   MOV     AH,07H          ; Function Call - Keyboard Input
   INT     21H             ; Issue request to DOS
   MOV     ExtChar,AL      ; Save extended character
Normal_Char:

or

   MOV     AH,06H          ; Function Call - Direct Console I/O
   MOV     DL,Char         ; Get character to display (not 0FFH)
   INT     21H             ; Issue request to DOS

-----

Character    LABEL  WORD ; Complete character
Char          DB      ?  ; Character buffer
ExtChar       DB      ?  ; Character buffer
```

Comments

If DL is FFH, AL returns with the 0 flag clear and an input character from the standard input device, if one is ready. If a character is not ready, the 0 flag will be set.

If DL is not FFH, DL is assumed to have a valid character that is output to the standard output device. This function does not check for Ctrl-Break, or Ctrl-PrtSc.

For function call 06H, extended ASCII codes require two function calls. The first call returns 00H as an indicator that the next call will return an extended code.

Purpose

Waits for a character to be read at the standard input device (unless one is ready), then returns the character in AL.

Example

```
  MOV   AH,07H        ; Function Call - Direct Console Input (no echo)
  INT   21H           ; Issue request to DOS
  MOV   Char,AL       ; Save character
  CMP   AL,0          ; Extended character ?
  JNE   Normal_Char   ; No!
  MOV   AH,07H        ; Function Call - Direct Console Input (no echo)
  INT   21H           ; Issue request to DOS
  MOV   ExtChar,AL    ; Save extended character
Normal_Char:

  -----

  Character     LABEL   WORD    ; Complete character
  Char          DB      ?       ; Character buffer
  ExtChar       DB      ?       ; Character buffer
```

Comments

As with function call 06H, no checks are made on the character.

For function call 07H, extended ASCII codes require two function calls. The first call returns 00H as an indicator that the next call will return an extended code.

08H —
Console Input Without Echo

Purpose

Waits for a character to be read at the standard input device (unless one is ready) and returns the character in AL.

Example

```
 MOV  AH,08H     ; Function Call - Console Input (no echo)
 INT  21H        ; Issue request to DOS
 MOV  Char,AL    ; Save character
 CMP  AL,0       ; Extended character ?
 JNE  Normal_Char; No!
 MOV  AH,08H     ; Function Call - Console Input (no echo)
 INT  21H        ; Issue request to DOS
 MOV  ExtChar,AL ; Save extended character
Normal_Char:
  -----

 Character     LABEL   WORD    ; Complete character
 Char          DB      ?       ; Character buffer
 ExtChar       DB      ?       ; Character buffer
```

Comments

The character is checked for Ctrl-Break. If Ctrl-Break is detected, an interrupt 23H is executed.

For function call 08H, extended ASCII codes require two function calls. The first call returns 00H as an indicator that the next call will return an extended code.

Purpose

Sends the characters in the string to the standard output device.

Example

```
MOV  AX,SEG String
MOV  DS,AX              ;Set DS:DX to string
MOV  DX,OFFSET String
MOV  AH,09H             ;Function Call - Display String
INT  21H                ;Issue request to DOS

 -----

String        DB        "This string ends at the first Dollar"
              DB        0DH,0AH
              DB        "$"
```

Comments

The character string in memory must be terminated by a **$** (24H). Each character in the string is output to the standard output device in the same form as function call 02H.

ASCII codes 0DH and 0AH represent carriage return and line feed, respectively.

0AH —
Buffered Keyboard Input

Purpose

Reads characters from the standard input device and places them in the buffer beginning at the third byte.

Example

```
MOV  AX,SEG Buffer
MOV  DS,AX              ;Set DS:DX to return Buffer
MOV  DX,OFFSET Buffer
MOV  AH,0AH             ;Function Call-Buffered
                        ;Keyboard Input
INT 21H                 ;Issue request to DOS

------

Buffer   DB   128         ; Max length of input
CurLen   DB   ?           ; Number of characters input
                          ; (excludes Return (0DH))
CurText  DB   128 DUP(?) ; Up to 128 characters allowed
```

Comments

The first byte of the input buffer specifies the number of characters the buffer can hold. This value cannot be 0. Reading the standard input device and filling the buffer continues until Enter is read. If the buffer fills to one less than the maximum number of characters it can hold, each additional character read is ignored and causes the bell to ring, until Enter is read. The second byte of the buffer is set to the number of characters received, excluding the carriage return (0DH), which is always the last character.

Purpose

Checks if there is a character available from the standard input device.

Example

```
In_LOOP:
  MOV     AH,0BH     ; Function Call - Check Input
  INT     21H        ; Issue request to DOS
  CMP     AL,-1      ; 0FFH indicates character available
  JNE     In_LOOP:
```

Comments

If a character is available from the STDIN device, AL is FFH. Otherwise, AL is undefined. If a Ctrl-Break is detected, an interrupt 23H is executed.

0CH —
Clear Keyboard Buffer and Invoke a Keyboard Function

Purpose

Clears the standard input device of any characters, then executes the function call number in AL.

Example

```
MOV     AH,0CH          ; Function Call - Clear keyboard &
                        ; Invoke function
MOV     AL,Function     ; Function Call to execute
                        ; (only 01H, 06H, 07H, 08H, and 0AH are allowed).
INT     21H             ; Issue request to DOS
                        ; Output depends on Function Call selected
```

Purpose

Writes to the disk file buffers that have been modified. All buffers are then made available for reuse.

Example

```
MOV     AH,0DH          ; Function Call - Disk Reset
INT     21H             ; Issue request to DOS
                        ; No return
```

Comments

It is necessary to close or commit all open files to correctly update the disk directory.

0EH —
Select Disk

Purpose

Selects the drive specified in DL (0=A, 1=B, etc.) (if valid) as the default drive.

Example

```
MOV     AH,0EH          ; Function Call - Select Disk
MOV     DL,Drive        ; Drive to select       (0=A:, 1=B:, ...)
INT     21H             ; Issue request to DOS
MOV     LastDrive,AL    ; Save max drive number (1=A:, 2=B:, ...)
MOV     AH,19H          ; Function Call - Get Current Disk
INT     21H             ; Issue request to DOS
CMP     AL,DL           ; Selected drive = requested
JNE     Error           ; No, Error!

-----

Drive         DB        ; New Drive to select
LastDrive     DB        ; Highest Valid Drive
```

Comments

The total number of unique drive letters, including diskette and fixed disk drives, that can be referenced is returned in AL. The value in AL is equal to the value of LASTDRIVE in CONFIG.SYS or the total number of installed devices, whichever is greater. For DOS 4.00 5 is the minimum value returned in AL. If the system has only one diskette drive, it is counted as two to be consistent with the philosophy of thinking of the system as having logical drives A and B.

Purpose

Searches the current directory for the named file and AL returns FFH if it is not found. If the named file is found, AL returns 00H and the FCB is filled as described below.

Example

```
MOV  AX,SEG FCB     ;Address FCB Parameter Block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,0FH         ;Function Call-FCB Open
INT  21H            ;Issue request to DOS

 -----

 FCB          LABEL   BYTE
 Drive        DB      0               ; Drive (0=Current, 1=A, 2=B, ...)
 FName        DB      "FILENAME"      ; File Name      (blank padded)
 Ext          DB      "EXT"           ; File Extension (blank padded)
              DB      25 DUP(0)       ; Filled in by DOS 4.00
```

Comments

AL is 00H if the file is opened.

AL is FFH if the file was not opened.

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

If the drive code was 0 (default drive), it is changed to the actual drive used (1 = A, 2 = B, and so on). This allows changing the default drive without interfering with subsequent operations on this file. The current block field (FCB bytes C-D) is set to 0. The size of the record to be worked with (FCB bytes E-F) is set to the system default of 80H. The size of the file and the date are set in the FCB from information obtained from the directory. You can change the default value for the

record size (FCB bytes E-F) or set the random record size and/or current record field. Perform these actions after the open, but before any disk operations.

The file is opened in compatibility mode. For information on compatibility mode, refer to function call 3DH.

Purpose

Closes a file.

Example

```
MOV  AX,SEG FCB          ; Address FCB Parameter Block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,10H              ; Function Call-FCB Close
INT  21H                 ; Issue request to DOS
CMP  AL,0                ; File Closed?
JNE  Error               ; No, Error!

-----

FCB           LABEL   BYTE
; Contents set by previous operations
```

Comments

AL is 00H if the file is closed.

AL is FFH if the file was not closed.

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

This function call must be executed on open files after file writes, and we highly recommend that it be used on all files. If the file is not found in its correct position in the current directory, it is assumed the disk was changed and AL returns FFH. Otherwise, the directory is updated to reflect the status in the FCB, the buffers for that file are flushed, and AL returns 00H.

11H —
Search for First Entry

Purpose

Searches the current directory for the first matching filename.

Example

```
MOV  AX,SEG DTA     ; Address Buffer for found file
MOV  DS,AX          ; information
MOV  DX,OFFSET DTA
MOV  AH,1AH         ; Function Call-Set DTA address
INT  21H            ; Issue request to DOS
MOV  AX,SEG FCB     ; Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,11H         ; Function Call-FCB search first
INT  21H            ; Issue request to DOS
CMP  AL,0           ; File found?
JNE  Error          ; No, Error!

-----

FCB     LABEL   BYTE
Fdrive  DB      0               ; Drive (0=Current, 1=A, 2=B, ...)
Fname   DB      "FILENAME"      ; File name      (blank padded, may use ?)
Fext    DB      "EXT"           ; File extension (blank padded, may use ?)
        DB      25 DUP(0)       ; Filled in by DOS

DTA     LABEL   BYTE
Ddrive  DB      ?               ; Drive
Dname   DB      "????????"      ; File Name      (blank padded)
Dext    DB      "???"           ; File Extension (blank padded)
        DB      25 DUP(0)       ; Filled in by DOS 4.00
```

Comments

AL is 00H if the file is found.

AL is FFH if the file was not found.

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

The current disk directory is searched for the first matching filename. If none are found, AL returns FFH. Global filename characters are allowed in the filename and extension. If a matching filename is

found, AL returns 00H and the locations at the disk transfer address are set as follows:

- If the FCB provided for searching was an extended FCB, the first byte at the disk transfer address is set to FFH followed by 5 bytes of 0, then the attribute byte from the search FCB, then the drive number used (1 = A, 2 = B, 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.
- If the FCB provided for searching was a standard FCB, then the first byte is set to the drive number used (1 = A, 2 = B), and the next 32 bytes contain the matching directory entry. Thus, the disk transfer address contains a valid unopened normal FCB.

Notes:

If an extended FCB is used, the following search pattern is used:

1. If the attribute is 0, only normal file entries are found. Entries for volume label, sub-directories, hidden and system files, are not returned.
2. If the attribute field is set for hidden or system files, or directory entries, it is an inclusive search. All normal file entries, plus all entries matching the specified attributes, are returned. To look at all directory entries except the volume label, the attribute byte may be set to hidden + system + directory (all 3 bits on).
3. If the attribute field is set for the volume label, it is considered an exclusive search, and *only* the volume label entry is returned.

12H —
Search for Next Entry

Purpose

Searches the current directory for the next matching filename.

Example

```
MOV       AX,SEG DTA    ; Address Buffer for found file
MOV       DS,AX         ; Information
MOV       DX,OFFSET DTA
MOV       AH,1AH        ; Function Call-Set DTA address
INT       21H           ; Issue request to DOS
MOV       AX,SEG FCB    ; Address FCB Parameter Block
MOV       DS,AX
MOV       DX,OFFSET FCB
MOV       AH,12H        ; Function Call-FCB Search Next
INT       21H           ; Issue request to DOS
CMP       AL,0          ; File found?
JNE     Error           ; No, Error!

-----

FCB          LABEL   BYTE
; As set by FCB Search First

DTA          LABEL   BYTE
Drive        DB      ?              ; Drive
Fname        DB      "????????"     ; File Name      (blank padded)
Ext          DB      "???"          ; File Extension (blank padded)
             DB      25 DUP(0)      ; Filled in by DOS 4.00
```

Comments

AL is 00H if the file is found.

AL is FFH if the file was not found.

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

After a matching filename has been found using function call 11H, function 12H may be called to find the next match to an ambiguous request.

The DTA contains information from the previous Search First or Search Next. All of the FCB, except for the name/extension field, is used to keep information necessary for continuing the search, so no disk operations may be performed if this FCB is between a previous function 11H or 12H call and this one.

13H —
Delete File

Purpose

Deletes all current directory entries that match the specified filename. The specified filename cannot be read-only.

Example

```
MOV  AX,SEG FCB       ;Address FCB Parameter Block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,13H           ;Function Call-FCB Delete File
INT  21H              ;Issue request to DOS
CMP  AL,0             ;File(s) Deleted?
JNE  Error            ;No, Error!

-----

FCB          LABEL   BYTE
Drive        DB      0              ; Drive
FName        DB      Filename       ; File Name      (blank padded, may use ?)
Ext          DB      Ext            ; File Extension (blank padded, may use ?)
             DB      25 DUP(0)      ; Filled in by DOS
```

Comments

AL is 00H if the file is found.

AL is FFH if the file was not found.

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

All matching current directory entries are deleted. The global filename character "?" is allowed in the filename or extension. If no directory entries match, AL returns FFH; otherwise AL returns 00H.

If the file is specified in read-only mode, the file is not deleted.

Note: Close open files before deleting them.

Network Access Rights: Requires Create access rights.

Purpose

Loads the record addressed by the current block (FCB bytes C-D) and the current record (FCB byte 1F) at the disk transfer address (DTA), then the record address is increased.

Example

```
MOV  AX,SEG DTA            ;Address Data buffer
MOV  DS,AX
MOV  DX,OFFSET DTA
MOV  AH,1AH                ;Function call Set DTA Address
INT  21H                   ;Issue request to DOS
MOV  AX,SEG FCB            ;Address FCB Parameter Block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,14H                ;Function Call-FCB Sequential Read
INT  21H                   ;Issue request to DOS
CMP  AL,0                  ;Data Read?
JNE  Error                 ;No, Error!

-----

FCB           LABEL   BYTE
; Set by previous open
DTA           LABEL   BYTE
              DB      ?Dup(0)   ;I/O buffer
```

Comments

AL is 00H if the read was successful.

AL is 01H if the file was at End of File (EOF).

AL is 02H if the read would have caused a wrap or overflow because the DTA was too small (the read was not completed).

AL is 03H if EOF (a partial record was read and filled out with 0's).

Use Function Call 59H (Get Extended Error) to determine the actual error condition. The length of the record is determined by the FCB record size field.

Network Access Rights: Requires Read access rights.

15H —
Sequential Write

Purpose

Writes the record addressed by the current block and record fields (size determined by the FCB record size field) from the disk transfer address. If records are less than the sector size, the record is buffered for an eventual write when a sector's worth of data is accumulated. Then the record address is increased.

Example

```
MOV  AX,SEG DTA          ;Address Data buffer
MOV  DS,AX
MOV  DX,OFFSET DTA
MOV  AH,1AH              ;Function Set DTA Address
INT  21H                 ;Issue request to DOS
MOV  AX,SEG FCB          ;Address FCB Parameter Block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,15H              ;Function Call-FCB Sequential Write
INT  21H                 ;Issue request to DOS
CMP  AL,0                ;Data Written?
JNE  Error               ;No, Error!

FCB    LABEL   BYTE
; Set by previous open
DTA    LABEL   BYTE
       DB      ?Dup(0)   ;I/O buffer
```

Comments

AL is 00H if the write was successful.

AL is 01H if the disk or diskette is full (write cancelled).

AL is 02H if the write would have caused a wrap or overflow because the DTA was too small (write cancelled).

Use Function Call 59H (Get Extended Error) to determine the actual error condition. If the file is specified in read-only mode, the sequential write is not performed and 01H is returned in AL.

Network Access Rights: Requires Write access rights.

Purpose

Creates a new file.

Example

```
MOV  AX,SEG FCB    ;Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,16H        ;Function Call-FCB create file
INT  21H           ;Issue request to DOS
CMP  AL,0          ;File created and opened?
JNE  Error         ;No, Error!

 -----

FCB             LABEL   BYTE
Fdrive          DB      0           ; Drive (0=Current, 1=A, 2=B, ...)
Fname           DB      "FILENAME"  ; File name      (blank padded)
Fext            DB      "EXT"       ; File extension (blank padded)
                DB      25 DUP(0)   ; Filled in by DOS
```

Comments

AL is 00H if the file is created and opened.

AL is FFH if the file was not created (normally a full directory or disk full).

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

If a matching entry is found it is reused. If no match is found, the directory is searched for an empty entry. If a match is found, the entry is initialized to a 0-length file, the file is opened (see function call 0FH), and AL returns 00H.

The file may be marked *hidden* during its creation by using an extended FCB containing the appropriate attribute byte.

Network Access Rights: Requires Create access rights.

17H —
Rename File

Purpose

Changes every matching occurrence of the first filename in the current directory of the specified drive to the second, with the restriction that two files cannot have the same name and extension.

Example

```
MOV  AX,SEG FCB       ;Address FCB Parameter Block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,17H           ;Function Call-FCB Rename File
INT  21H              ;Issue request to DOS
CMP  AL,0             ;File(s) Renamed?
JNE  Error            ;No, Error!

-----

FCB      LABEL   BYTE
Fdrive   DB      0           ; Drive (0=Current, 1=A, 2=B, ...)
Fname    DB      "FILENAME"  ; File Name          (blank padded, may use ?)
Fext     DB      "EXT"       ; File Extension     (blank padded, may use ?)
         DB      5 DUP(0)    ; Reserved
NewName  DB      "FILENAME"  ; New File Name      (blank padded, may use ?)
NewExt   DB      "EXT"       ; New File Extension (blank padded, may use ?)
         DB      7 DUP(0)    ; Reserved
```

Comments

AL is 00H if the file or files were renamed.

AL is FFH if a file in the current directory did not match or the new name already exists.

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

The modified FCB has a drive code and filename in the usual position, and a second filename on the sixth byte after the first (DS:DX+11H) in what is normally a reserved area.

If "?"s appear in the second name, the corresponding positions in the original name are unchanged.

If the file is specified in read-only mode, the file is not renamed.

Network Access Rights: Requires Create access rights.

19H —
Current Disk

Purpose

Determines the current default drive.

Example

```
        MOV     AH,19H    ; Function Call - Get Current Disk
        INT     21H       ; Issue request to DOS
        MOV     Disk,AL   ; Save Current Disk
  ----

Disk    DB      ?         ; Current Disk code (0=A:, 1=B:, ...)
```

Comments

AL returns with the code of the current default drive (0 = A, 1 = B, and others.).

1AH —
Set Disk Transfer Address

Purpose

Sets the disk transfer address to DS:DX.

Example

```
MOV  AX,SEG DTA        ;Address Buffer
MOV  DS,AX
MOV  DX,OFFSET DTA
MOV  AH,1AH            ;Function Call-Set DTA address
INT  21H               ;Issue request to DOS
 -----


DTA    LABEL   BYTE    ; Data Buffer
```

Comments

The area defined by this call is from the address in DS:DX to the end of the segment in DS. DOS 4.00 does not allow disk transfers to wrap around within the segment, or overflow into the next segment. If you do not set the DTA, the default DTA is offset 80H in the program segment prefix. To get the DTA, issue function call 2FH.

1BH —
Allocation Table Information

Purpose

Returns information about the allocation table for the default drive.

Example

```
MOV  AH,1BH                  ; Function Call - Allocation Table
                             ; Information
INT  21H                     ; Issue request to DOS
MOV  NumAllocUnits,DX        ; Save Number of Allocation Units
MOV  NumSectsAllocUnit,AL    ; Save Number of Sectors/Allocation Unit
MOV  SectSize,CX             ; Save of Sector Size
MOV  WORD PTR MediaType@+0,BX; Save Pointer to Media Type Byte
MOV  WORD PTR MediaType@+2,DS

-----

NumAllocUnits     DW     ?       ; Number of Allocation Units on Current Drive
NumSectsAllocUnit DB     ?       ; Number Sectors in an Allocation Unit
SectSize          DW     ?       ; Sector Size
MediaType@        DD     ?       ; Pointer to Media Type byte
```

Comments

Refer to function call 36H (Get Disk Free Space).

1CH — Allocation Table Information for Specific Device

Purpose

Returns allocation table information for a specific device.

Example

```
MOV   AH,1CH                    ; Function Call -
                                ; Allocation Table Information
MOV   DL,Drive                  ; Drive requested  (0=current,
                                ; 1=A:, ...)
INT   21H                       ; Issue request to DOS
MOV   NumAllocUnits,DX          ; Save Number of Allocation Units
MOV   NumSectsAllocUnit,AL      ; Save Number of Sectors/Allocation Unit
MOV   SectSize,CX               ; Save of Sector Size
MOV   WORD PTR MediaType@+0,BX ; Save Pointer to Media Type Byte
MOV   WORD PTR MediaType@+2,DS

-----
Drive             DB            ; drive number to get info for
NumAllocUnits     DW    ?       ; Number of Allocation Units
                                ; on specified drive
NumSectsAllocUnit DB    ?       ; Number Sectors in an
                                ; Allocation Unit
SectSize          DW    ?       ; Sector Size
MediaType@        DD    ?       ; Pointer to Media Type byte
```

Comments

This call is the same as call 1BH, except that on entry DL contains the number of the drive that contains the needed information (0 = default, 1 = A, and so forth.). For more information on DOS 4.00 disk allocation, refer to "The Disk Directory" on page 2-3. Also, refer to function call 36H (Get Disk Free Space).

21H —
Random Read

Purpose

Reads the record addressed by the current block and current record fields into memory at the current disk transfer address.

Example

```
                              ;Set up FCB
  MOV  AX,SEG DTA             ;Address Data buffer
  MOV  DS,AX
  MOV  DX,OFFSET DTA
  MOV  AH,1AH                 ;Function Set DTA Address
  INT  21H                    ;Issue request to DOS
  MOV  AX,SEG FCB             ;Address FCB Parameter Block
  MOV  DS,AX
  MOV  DX,OFFSET FCB
  MOV  AH,21H                 ;Function Call-FCB Random Read
  INT  21H                    ;Issue request to DOS
  CMP   AL,0                  ;Data Read?
  JNE   Error                 ;No, Error!

-----

FCB           LABEL   BYTE
; Set by previous open
; DTA label byte
```

Comments

AL is 00H if the read was successful.

AL is 01H if the file was at End of File (EOF) (no data read).

AL is 02H if the read would have caused a wrap or overflow because the DTA was too small (the read was not completed).

AL is 03H if EOF (a partial record was read and filled out with 0's).

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

The current block and current record fields are set to agree with the random record field. The record addressed by these fields is read

into memory at the current disk transfer address. For information on record size see Chapter 4, "Accessing Files Using File Control Blocks" on page 4-1.

Note: Function 24H must be called before using this function.

Network Access Rights: Requires Read access rights.

22H —
Random Write

Purpose

Writes the record addressed by the current block and current record fields from the current disk transfer address.

Example

```
;Set up FCB

MOV  AX,SEG FCB       ;Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,24H           ;Function Call-FCB Set
                      ;Relative record field
INT  21H              ;Issue request to DOS
MOV  AX,SEG DTA       ;Address data buffer
MOV  DS,AX
MOV  DX,OFFSET DTA
MOV  AH,1AH           ;Function Set DTA address
INT  21H              ;Issue request to DOS
MOV  AX,SEG FCB       ;Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,22H           ;Function Call-FCB random writers
INT  21H              ;Issue request to DOS
CMP  AL,0             ;Data written?
JNE  Error            ;No, error!

-----

FCB           LABEL   BYTE
; Set by previous open
DTA           LABEL   BYTE
```

Comments

AL is 00H if the write was successful.

AL is 01H if the write or diskette is full (write cancelled).

AL is 02H if the read would have caused a wrap or overflow because the DTA was too small (write cancelled).

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

The current block and current record fields are set to agree with the random record field. Then the record addressed by these fields is written (or in the case of records not the same as sector sizes — buffered) from the disk transfer address.

If the file is specified in read-only mode, the random write is not performed.

Network Access Rights: Requires Write access rights.

23H —
File Size

Purpose

Searches the current directory for an entry that matches the specified file and sets the FCBs random record field to the number of records in the file.

Example

```
MOV  AX,SEG FCB      ; Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,23H          ; Function Call-FCB file size
INT  21H             ; Issue request to DOS
CMP  AL,0            ; File found?
JNE  Error           ; No, error!

-----

FCB          LABEL   BYTE
Drive        DB      0              ; Drive (0=Current, 1=A, 2=B, ...)
Name         DB      "FILENAME"     ; File name      (blank padded)
Ext          DB      "EXT"          ; File extension (blank padded)
             DB      25 DUP(0)      ; Filled in by DOS
```

Comments

AL is 00H if the file exists.

AL is FFH if the file was not created (normally a full directory or disk full).

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

The directory is searched for the matching entry. If a matching entry is found, the random record field is set to the number of records in the file (in terms of the record size field rounded up). If no matching entry is found, AL returns FFH.

Note: If you do not set the FCB record size field before using this function, incorrect information is returned.

Purpose

Sets the random record field to the same file address as the current block and record fields.

Example

```
MOV  AX,SEG FCB     ;Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,24H         ;Function Call-FCB set
                    ;Relative record field
INT  21H            ;Issue request to DOS

-----

FCB           LABEL   BYTE
; Set by previous open
```

Comments

You must call this function before you perform random reads and writes, and random block reads and writes.

25H —
Set Interrupt Vector

Purpose
Sets the interrupt vector table for the interrupt number.

Example
```
MOV  AX,SEG Handler     ;Address new handler
MOV  DS,AX
MOV  DX,OFFSET Handler
MOV  AH,25H             ;Function Call - Set Interrupt
                        ;Vector
MOV  AL,Vector
INT  21H                ;Issue request to DOS

-----

Vector     DB     ?     ;Number of vector to be replaced
Handler:                ;Code to process interrupt
```

Comments
The interrupt vector table for the interrupt number specified in AL is set to address contained in DS:DX. Use function call 35H (Get Interrupt Vector) to obtain the contents of the interrupt vector.

26H — Create New Program Segment

Purpose

Creates a new program segment.

Example

```
MOV     AH,26H              ; Function Call - Create Program
                            ; Segment
MOV     DX,SEG PSP          ; Segment address to create new PSP
INT     21H                 ; Issue request to DOS

-----

PSP         LABEL   BYTE    ; Area to fill in
            DB      100H DUP(0)
```

Comments

The entire 100H area at location 0 in the current program segment is copied into location 0 in the new program segment. The memory size information at location 6 in the new segment is updated and the current termination, Ctrl-Break exit and critical error addresses from interrupt vector table entries for interrupts 22H, 23H, and 24H are saved in the new program segment starting at 0AH. They are restored from this area when the program ends.

Note: The EXEC function call 4BH provides a more complete service. Therefore, you should use the EXEC 4BH and avoid using this call.

27H —
Random Block Read

Purpose

Reads the specified number of records (in terms of the record size field) from the file address specified by the random record field into the disk transfer address.

Example

```
                        ;Set up disk transfer address

MOV  AX,SEG DTA         ;Address data buffer
MOV  DS,AX
MOV  DX,OFFSET DTA
MOV  AH,1AH             ;Function set DTA address
INT  21H                ;Issue request to DOS

MOV  AX,SEG FCB         ;Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,24H             ;Function Call - FCB Set
                        ;Relative Record Field
INT  21H                ;Issue request to DOS
MOV  AX,SEG FCB         ;Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  CX,Records_to_read ;number of records to read
MOV  AH,27H             ;Function Call - FCB random block read
INT  21H                ;Issue request to DOS
CMP  AL,0               ;Data read?
JNE  Error              ;No, error!

 -----

 FCB              LABEL   BYTE
 ; Set by previous open
 DTA              LABEL   BYTE
                  DB      ?Dup(0)     ;I/O buffer
 Records_to_read  DW      ?
```

Comments

AL is 00H if the read was successful.

AL is 01H if the file was at End of File (EOF) (no data read).

AL is 02H if the read would have caused a wrap or overflow because the DTA was too small (the read was not completed.).

AL is 03H if EOF (a partial record was read and filled out with zeros).

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

The random record field and the current block/record fields are set to address the next record (the first record not read).

Note: Function 24H must be called before using this function.

Network Access Rights: Requires Read access rights.

28H —
Random Block Write

Purpose

Writes the specified number of records from the disk transfer address into the file address specified by the random record field.

Example

```
                              ;Set up disk transfer address
MOV  AX,SEG DTA               ;Address data buffer
MOV  DS,AX
MOV  DX,OFFSET DTA
MOV  AH,1AH                   ;Function set DTA address
INT  21H                      ;Issue request to DOS

MOV  AX,SEG FCB               ;Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  AH,24H                   ;Function Call-FCB set
                              ;Relative record field
INT  21H                      ;Issue request to DOS


MOV  AX,SEG FCB               ;Address FCB parameter block
MOV  DS,AX
MOV  DX,OFFSET FCB
MOV  CX,Records_to_write      ;Number of records to write
MOV  AH,28H                   ;Function Call - FCB Random Block write
INT  21H                      ;Issue request to DOS
CMP  AL,0                     ;Data written?
JNE  Error                    ;No, error!

  -----

DTA     LABEL     BYTE

        DB        ?DUP(0)     ; I/O Buffer
```

Comments

AL is 00H if the write was successful.

AL is 01H if the disk or diskette is full (write cancelled).

AL is 02H if the write would have caused a wrap or overflow because the DTA was too small (write cancelled).

Use Function Call 59H (Get Extended Error) to determine the actual error condition.

If there is insufficient space on the disk, AL returns 01H and no records are written. If CX is 0 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.)

Note: Function call 24H must be called before using this function.

Network Access Rights: Requires Write access rights.

29H —
Parse Filename

Purpose
Parses the specified filename.

Example

```
MOV   AX,SEG CmdBuf
MOV   DS,AX            ;Address command string
MOV   SI,OFFSET CmdBuf
MOV   AX,SEG FCB
MOV   ES,AX            ;Address FCB Parameter Block
MOV   DI,OFFSET FCB
MOV   AH,29H           ;Function Call - FCB Parse Filename
MOV   AL,OPTIONS       ;Set desired action
INT   21H              ;Issue request to DOS

CMP   AL,-1            ;Drive valid?
JE    Error            ;No, Error!
-----

CmdBuf      LABEL    BYTE
            DB       " a:file.ext      ",0DH

FCB         LABEL    BYTE
; Created in a pre-open state based on input found.
  Options   DB   ?   ;parsing options
```

Comments
The contents of AL are used to determine the action to take, as shown below:

```
      <must = 0>
bit:  7   6   5   4   3   2   1   0
```

If bit 0 = 1, leading separators are scanned off the command line at DS:SI. Otherwise, no scan-off of leading separators takes place.

If bit 1 = 1, the drive ID byte in the FCB will be set (changed) *only* if a drive was specified in the command line being parsed.

If bit 2 = 1, the filename in the FCB will be changed only if the command line contains a filename.

If bit 3 = 1, the filename extension in the FCB will be changed only if the command line contains a filename extension.

Filename separators include the following characters:

: . : , = + along with TAB and SPACE. Filename terminators include all of these characters plus ,<, >, |, /, ", [,], and any control characters.

Output:

AL is 00H if no global characters (? or *) were found in the Command String.

AL is 01H if global characters (? or *) were found in the Command String.

AL is FFH if the drive specified is invalid.

The command line is parsed for a filename of the form *d:filename.ext*, and if found, a corresponding unopened FCB is created at ES:DI. If no drive specifier is present, it is assumed to be all blanks. If the character * appears in the filename or extension, it and all remaining characters in the name or extension are set to ?.

DS:SI returns pointing to the first character after the filename and ES:DI points to the first byte of the formatted FCB. If no valid filename is present, ES:DI+1 contains a blank.

2AH —
Get Date

Purpose

Returns the day of the week, the year, month and date.

Example

```
        MOV     AH,2AH          ; Function Call - Get Date
        INT     21H             ; Issue request to DOS
        MOV     DayofWeek,AL    ; Save Day of the Week
        MOV     Year,CX         ; Save Year
        MOV     Month,DH        ; Save Month
        MOV     Day,DL          ; Save Day

-----

DayofWeek       DB      ?       ; 0=Sunday, ... 6=Saturday
Year            DW      ?       ; 1980 to 2099
Month           DB      ?       ; 1 to 12
Day             DB      ?       ; 1 to 31
```

Comments

If the time-of-day clock rolls over to the next day, the date is adjusted accordingly, taking into account the number of days in each month and leap years.

Purpose

Sets the date (also sets CMOS clock, if present).

Example

```
        MOV     AH,2BH          ; Function Call - Set Date
        MOV     CX,Year         ; Set Year
        MOV     DH,Month        ; Set Month
        MOV     DL,Day          ; Set Day
        INT     21H             ; Issue request to DOS
        CMP     AL,0            ; Valid Date?
        JNE     Error           ; No!
-----

Year            DW      ?       ; 1980 to 2099
Month           DB      ?       ; 1 to 12
Day             DB      ?       ; 1 to 31
```

Comments

AL is 00H if the date is valid and the operation is successful.

AL is FFH if the date is not valid.

On entry, CX:DX must have a valid date in the same format as returned by function call 2AH.

On return, AL returns 00H if the date is valid and the set operation is successful. AL returns FFH if the date is not valid.

2CH —
Get Time

Purpose

Returns the time; hours, minutes, seconds and hundredths of seconds.

Example

```
        MOV     AH,2CH          ; Function Call -
                                ; Get Time
        INT     21H             ; Issue request to DOS
        MOV     Hour,CH         ; Save Hour
        MOV     Minute,CL       ; Save Minute
        MOV     Second,DH       ; Save Second
        MOV     Hundredth,DL    ; Save Partial Second

-----

Hour            DB      ?       ; 0 to 23
Minute          DB      ?       ; 0 to 59
Second          DB      ?       ; 0 to 59
Hundredth       DB      ?       ; 0 to 99
```

Comments

On entry, AH contains 2CH. On return, CX:DX contains the time-of-day. Time is actually represented as four 8-bit binary quantities as follows:

CH	Hours (0—23)
CL	Minutes (0—59)
DH	Seconds (0—59)
DL	1/100 seconds (0—99).

This format is readily converted to a printable form yet can also be used for calculations, such as subtracting one time value from another.

Purpose

Sets the time (also sets the CMOS clock, if present).

Example

```
        MOV     AH,2DH          ; Function Call - Set Time
        MOV     CH,Hour         ; Sst Hour
        MOV     CL,Minute       ; Set Minute
        MOV     DH,Second       ; Set Second
        MOV     DL,Hundredth    ; Set Partial Second
        INT     21H             ; Issue request to DOS
        CMP     AL,0            ; Valid Time?
        JNE     Error           ; No!

-----

Hour        DB      ?       ; 0 to 23
Minute      DB      ?       ; 0 to 59
Second      DB      ?       ; 0 to 59
Hundredth   DB      ?       ; 0 to 99
```

Comments

AL is 00H if the time is valid.

AL is FFH if the time is not valid.

On entry, CX:DX has time in the same format as returned by function 2CH. On return, if any component of the time is not valid, the set operation is cancelled and AL returns FFH. If the time is valid, AL returns 00H.

If your system has a CMOS realtime clock, it will be set.

2EH —
Set/Reset Verify Switch

Purpose
Sets the verify switch.

Example

```
; To set VERIFY=OFF

     MOV     AH,2EH          ; Function Call - Set
                             ; VERIFY
     MOV     AL,0            ; Set OFF
     INT     21H             ; Issue request to DOS

; To set VERIFY=ON

     MOV     AH,2EH          ; Function Call - Set
                             ; VERIFY
     MOV     AL,1            ; Set ON
     INT     21H             ; Issue request to DOS
```

Comments

On entry, AL must contain 01H to turn verify on, or 00H to turn verify off. When verify is on, DOS 4.00 performs a verify operation each time it performs a disk write to assure proper data recording. Although disk recording errors are very rare, this function has been provided for applications in which you may wish to verify the proper recording of critical data. You can obtain the current setting of the verify switch through function call 54H.

Note: Verification is not supported on data written to a network disk.

2FH —
Get Disk Transfer Address (DTA)

Purpose
Returns the current disk transfer address.

Example
```
        MOV     AH,2FH                  ; Function Call - Get
                                        ; DTA Address
        INT     21H                     ; Issue request to DOS
        MOV     WORD PTR DTA@+0,BX      ; Save Address
        MOV     WORD PTR DTA@+2,ES

-----

DTA@            DD      ?               ; DTA Buffer
```

Comments
On entry, AH contains 2FH. On return, ES:BX contains the current Disk Transfer Address. You can set the DTA using function call 1AH.

30H —
Get DOS Version Number

Purpose
Returns the DOS version number.

Example

```
        PUSH    CX              ; CX and BX destroyed in call
        PUSH    BX
        MOV     AH,30H          ; Function Call - Get DOS 4.00
                                ; Version
        INT     21H             ; Issue request to DOS
        MOV     MajorVersion,AL ; Save Version
        MOV     MinorVersion,AH
        POP     BX
        POP     CX

-----
MajorVersion  DB      ?         ; X of X.YY
MinorVersion  DB      ?         ; YY of X.YY
```

Comments

On entry, AH contains 30H. On return, BX and CX are set to 0. AL contains the major version number. AH contains the minor version number.

If AL returns a major version number of 0, you can assume that the DOS version is pre-DOS 2.00.

31H — Terminate Process and Remain Resident

Purpose

Terminates the current process and attempts to set the initial allocation block to the memory size in paragraphs.

Example

```
        MOV     AH,31H          ; Function Call - Terminate
                                ; and Keep Process
        MOV     AL,RetCode      ; Set value of ERRORLEVEL
        MOV     DX,MySize       ; Set my program and data size
        INT     21H             ; Issue request to DOS
        INT     20H             ; Be safe if on DOS 4.00 Version 1.X

-----

RetCode         DB      ?       ; Value to return to my EXEC'er
MySize          DW      ?       ; Size of my code and data
                                ; (in paragraphs)
```

Comments

On entry, AL contains a binary return code. DX contains the memory size value in paragraphs. This function call does not free up any other allocation blocks belonging to that process. Files opened by the process are not closed when the call is executed. The return code passed in AL is retrievable by the parent through Wait (function call 4DH) and can be tested through the ERRORLEVEL batch subcommands.

Memory is used efficiently if the block containing a copy of the environment is deallocated before terminating. This can be done by loading ES with the segment contained in 2C of the PSP, and issuing function call 49H (Free Allocated Memory). The five standard handles, 0000 through 0004, should be closed before exiting.

33H —
Get/Set System Value

Purpose

Set or get the state of System Values such as BREAK (Ctrl-Break checking).

Example

```
; To check BREAK state

      MOV     AH,33H          ; Function Call - Get/Set
                              ; System value
      MOV     AL,0            ; Do Get BREAK
      INT     21H             ; Issue request to DOS
      MOV     BREAK,DL        ; Save state

; To set BREAK=OFF

      MOV     AH,33H          ; Function Call - Get/Set
                              ; System value
      MOV     AL,1            ; Do Set BREAK
      MOV     DL,0            ; Set OFF
      INT     21H             ; Issue request to DOS

; To set BREAK=ON

      MOV     AH,33H          ; Function Call - Get/Set
                              ; System Value
      MOV     AL,1            ; Do Set BREAK
      MOV     DL,1            ; Set ON
      INT     21H             ; Issue request to DOS

; To get the Boot Drive

      MOV     AH,33H          ; Function Call - Get/Set
                              ; System Value
      MOV     AL,5            ; Do Get Boot Drive
      INT     21H             ; Issue request to DOS
      MOV     Drive,DL        ; Save boot drive

----

BREAK         DB      ?       ; Current BREAK state (0=OFF, 1=ON)
Drive         DB      ?       ; DOS 4.00 boot drive
```

Comments

For BREAK:

AL contains 00H to request the current state of Ctrl-Break checking. On return DL contains the current state (00H = OFF, 01H = ON).

AL contains 01H to set the state. DL contains the new state (00H = OFF, 01H = ON).

For boot drive:

AL contains 05H to request the boot drive. On return DL contains the drive (A:=1, C:=3, ...).

35H —
Get Interrupt Vector

Purpose
To obtain the address in an interrupt vector.

Example
```
        MOV   AH,35H                ; Function Call -
                                    ; Set Interrupt Vector
        MOV   AL,Vector             ; Vector to get (0 to 255)
        INT   21H                   ; Issue request to DOS
        MOV   WORD PTR OldVect+0,BX
        MOV   WORD PTR OldVect+2,ES

-----

OldVect       DD      ?             ; Previous vector contents
Vector        DB      ?             ; Vector number to get
```

Comments
On entry, AH contains 35H. AL contains a hexadecimal interrupt number. On return, ES:BX contains the CS:IP interrupt vector for the specified interrupt. Use function call 25H (Set Interrupt Vector) to set the interrupt vectors.

Purpose

Returns the disk free space (available clusters, clusters/drive, bytes/sector).

Example

```
        MOV     AH,36H                  ; Function Call -
                                        ; Get disk free space
        MOV     DL,Drive                ; Drive to query
                                        ; (0=current, 1=A:,
                                        ; 2=B:, ...)
        INT     21H                     ; Issue request to DOS
        CMP     AX,-1                   ; Error?
        JE      Error                   ; Yes
        MOV     SectAU,AX               ; Save allocation unit
                                        ; Size
        MOV     AvailAU,BX              ; Save free allocation
                                        ;  Units
        MOV     SectSize,CX             ; Save sector size
        MOV     TotalAU,DX              ; Save disk size


        MOV     AX,SectSize             ; Calculate allocation
                                        ; Unit size
        MUL     SectAU
        MOV     CX,AX                   ; CX = bytes/AU
        MOV     AX,TotalAU              ; Calculate total space
        MUL     CX
        MOV     WORD PTR TotalBytes+0,AX ; Save it
        MOV     WORD PTR TotalBytes+2,DX
        MOV     AX,AvailAU              ; Calculate free space
        MUL     CX
        MOV     WORD PTR FreeBytes+0,AX  ; Save it
        MOV     WORD PTR FreeBytes+2,DX

-----

SectAU          DW      ?               ; Sectors in an
                                        ; Allocation unit
AvailAU         DW      ?               ; Free allocation units
SectSize        DW      ?               ; Bytes in a sector
TotalAU         DW      ?               ; Number of allocation
                                        ; Units on DL disk
TotalBytes      DD      ?               ; Disk size in bytes
FreeBytes       DD      ?               ; Free space in bytes
Drive           DD      ?               ; Drive number to get info for
```

36H —
Get Disk Free Space

Comments

If the drive number in DL was valid, BX contains the number of available allocation units, DX contains the total number of allocation units on the drive, CX contains the number of bytes per sector, and AX contains the number of sectors for each allocation unit.

38H —
Get or Set Country Dependent Information

Purpose

Sets the Active Country or returns country dependent information.

Example

```
; To set the Current Country

        MOV     AH,38H          ; Function Call - Get/Set
                                ; Country Information
        MOV     AL,CountryID    ; Country ID (-1 if >= 255)
        MOV     BX,CountryIDX   ; Country ID (if AL=-1)
        MOV     DX,-1           ; Indicate set country code
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

; To get Country Information

        MOV     AX,SEG Buffer
        MOV     DS,AX
        MOV     DX,OFFSET Buffer
        MOV     AH,38H
        MOV     AL,CountryID    ; Country ID (-1 if >= 255)
                                ; (0 to get current country)
        MOV     BX,CountryIDX   ; Country ID (if AL=-1)

        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     CountryCode,BX  ; Save current Country Code

----

CountryCode     DW      ?       ; Current country code

CountryIDX      DW      ?       ; Extended country code for input

Buffer          LABEL   WORD    ; Country information (see format below)
CountryID       DB      ?       ; Country code for input
```

38H —
Get or Set Country Dependent Information

Country Information

```
DateFormat      DW      ?           ; Date Format:
                                    ;   0 = m d y order
                                    ;   1 = d m y order
                                    ;   2 = y m d order
$Symbol         DB      "????",0    ; Currency Symbol
                                    ;   example: "DM",0,?,?
Sep1000         DB      "?",0       ; Thousands Separator
                                    ;   example: ",",0
Sep1            DB      "?",0       ; Fractions Separator
                                    ;   example: ".",0
SepDate         DB      "?",0       ; Date Separator
                                    ;   example: "/",0
SepTime         DB      "?",0       ; Time Separator
                                    ;   example: ":",0
$Format         DB      ?           ; Currency Format:
                                    ;   0 = currency symbol, value
                                    ;   1 = value, currency symbol
                                    ;   2 = currency symbol, space, value
                                    ;   3 = value, space, currency symbol
                                    ;   4 = currency symbol is decimal separator
SigDigits       DB      ?           ; Number of Significant Digits in Currency
TimeFormat      DB      ?           ; Time Format:
                                    ;   0 = 12 hour clock
                                    ;   1 = 24 hour clock
UpperCaseAL@    DD      ?           ; Address of Routine to Upper Case AL
                                    ;   Only for values >=80H
SepData         DB      "?",0       ; Data List Separator
                                    ;   example: ",",0
Reserved        DW      5 DUP(?)    ; Reserved for future
```

Comments

The date format has the following values and meaning:

Code	Date
0 = USA	m d y
1 = Europe	d m y
2 = Japan	y m d

Case Map Call Address: The register contents for the case map call are:

On Entry	Register Contents
AL	ASCII code of character to be converted to uppercase

On Return	Register Contents
AL	ASCII code of the uppercase input character

The case map call address is in a form suitable for a FAR call indirect.

Returns

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Function Call 65H, (Get Extended Country Information), returns more country information and is preferred.

Setting the Current County Code by using this function call is not recommended. The user can set it by placing a COUNTRY command in the CONFIG.SYS file. The Country Code set by the user should not be changed. The NLSFUNC DOS 4.00 extension must be installed to change the Current Country.

39H —
Create Subdirectory (MKDIR)

Purpose
Creates the specified directory.

Example
```
MOV  AX,SEG    DName  ;Directory Name
MOV  DS,AX
MOV  DX,OFFSET DName
MOV  AH,39H           ;Function-Make a directory
INT  21H              ;Issue request to DOS
JC   Error

 ----

DName          DB      "?? .. ??",0  ; ASCIIZ Name
                                     ; Example:
                                     ; "c:\dir",0
```

Comments
On entry, DS:DX contains the address of an ASCIIZ string with drive and directory path names. All directory levels other than the last one in the name must exist before using this function. Only one directory level at a time can be created with this function. The maximum length of the ASCIIZ string is 64 characters.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Network Access Rights: Requires Create access rights.

Purpose

Removes the specified directory.

Example

```
MOV  AX,SEG DName              ;Directory name
MOV  DS,AX
MOV  DX,OFFSET DName
MOV  AH,3AH                    ;Function-Remove directory
INT  21H                       ;Issue request to DOS
JC   Error

 ----

DName          DB      "?? .. ??",0  ; ASCIIZ Name
                                     ;   example: "c:\dir",0
```

Comments

On entry, DS:DX contains the address of an ASCIIZ string with the drive and directory path names. The specified directory is removed from the structure. The current directory or a directory with files in it cannot be removed.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Network Access Rights: Requires Create access rights.

3BH —
Change the Current Directory (CHDIR)

Purpose
Changes the current directory to the specified directory.

Example
```
MOV  AX,SEG DName              ;Directory name
MOV  DS,AX
MOV  DX,OFFSET DName
MOV  AH,3BH                    ;Function - Change directory
INT  21H                       ;Issue request to DOS
JC   Error

 ----

DName          DB      "?? .. ??",0  ; ASCIIZ Name
                                     ;   example: "c:\dir",0
```

Comments
On entry, DS:DX contains the address of an ASCIIZ string with drive and directory path names. The string is limited to 64 characters and cannot contain a network path. If any member of the directory path does not exist, the directory path is not changed. Otherwise, the current directory is set to the ASCIIZ string.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Purpose

Creates a new file or shortens an old file to 0 length in preparation for writing.

Example

```
        MOV     AX,SEG FName            ;File name
        MOV     DS,AX
        MOV     DX,OFFSET FName
        MOV     AH,3CH                  ;Function - Create a File
        MOV     CX,Attribute            ; Attribute of the file
                                        ; Allowed values
                                        ;  0001H=Read only
                                        ;  0002H=Hidden
                                        ;  0004H=System
                                        ;  0008H=Volume label
        INT     21H                     ; Issue request to DOS
        JC      Error                   ; Error code in AX
        MOV     Handle,AX               ; Save file handle for

  ----

  FName         DB      "?? .. ??",0  ; ASCIIZ Name
                                        ;   example: "c:\dir\file.ext",0
  Handle        DW      ?             ; File handle
  Attribute     DW      ?             ; Attributes for directory entry
```

Comments

If the file did not exist, the file is created in the appropriate directory and the file is given the read/write access code. The file is opened for read/write, the read/write pointer is set to the first byte of the file and the handle is returned in AX. Note that function call 43H (Change File Mode) can be used later to change the file's attribute.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

This function does not replace an existing volume label. You must delete the existing volume label before issuing this call.

Network Access Rights: Requires Create access rights.

3DH —
Open a File

Purpose
Opens the specified file.

Example

```
MOV  AX,SEG FName          ; File name
MOV  DS,AX
MOV  DX,OFFSET FName
MOV  AH,3DH                ; Function — Open a File
MOV  AL,OpenMode
INT  21H                   ; Issue request to DOS
JC   Error                 ; Error code in AX
MOV  Handle,AX             ; Save file handle for following operations

 ----

 FName         DB      "?? .. ??",0  ; ASCIIZ Name
                                     ;   example: "c:\dir\file.ext",0
 Handle        DW      ?             ; File Handle
 OpenMode      DB      ?             ; Open mode
```

Comments

The read/write pointer is set at the first byte of the file and the record size of the file is 1 byte. The read/write pointer can be changed with function call 42H. The returned file handle must be used for subsequent input and output to the file. The file's date and time can be obtained or set through call 57H, and its attribute can be obtained through call 43H.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Notes:

1. This call opens any normal or hidden file whose name matches the name specified.
2. Device names cannot end in a colon.
3. When a file is closed, any sharing restrictions placed on it by the open are canceled.

4. File sharing must be loaded, or the file must be a network file for the sharing modes to function. Refer to the SHARE command.
5. The file read-only attribute can be set when creating the file using extended FCBs or specifying the appropriate attribute in CX and using the CHMOD interrupt 21H function call or the DOS 4.00 ATTRIB command.
6. If the file is inherited by the subordinate process, all sharing and access restrictions are also inherited.
7. If an open file handle is duplicated by either of the DUP function calls, all sharing and access restrictions are also duplicated.

Open Mode

The open mode is defined in AL and consists of four bit-oriented fields:

Inheritance flag	Specifies if the opened file is inherited by a subordinate process.
Sharing mode field	Defines which operations other processes can perform on the file.
Reserved field	
Access field	Defines which operations the current process can perform on the file.

Bit Fields

The bit fields are mapped as follows:

```
                <I> < S > <R> < A >
Open Mode bits   7  6 5 4  3  2 1 0
```

I Inheritance flag

If I = 0; File is inherited by subordinate processes.
If I = 1; File is private to the current process.

S Sharing Mode

The file is opened as follows:

S = 000 — Compatibility mode
S = 001 — DenyRead/Write mode (exclusive)

S = 010 — DenyWrite mode
S = 011 — DenyRead mode
S = 100 — DenyNone mode

Any other combinations are invalid.

When opening a file, you must inform DOS 4.00 which operations any other processes, in sharing mode, can perform on the file.

The default, compatibility mode, denies all other computers in a network access to the file. If other processes can continue to read the file while your process is operating on it, specify DenyWrite. DenyWrite prohibits writing by other processes, but allows reading.

Similarly, you must specify which operations, or access modes, your process can perform. The default access mode, ReadWrite, causes the open request to fail if another process on the computer or any other computer on a network has the file opened with any sharing mode other than DenyNone. If you intend to read from the file only, your Open will succeed unless all other processes have specified DenyNone or DenyWrite. File sharing requires cooperation of both sharing processes.

R Reserved (set this bit field to 0).

A Access

The file access is assigned as follows:

If A = 000; Read access
If A = 001; Write access
If A = 010; Read/Write access

Any other combinations are invalid.

Network Access Rights: If the Access field (A) of the Open mode field (AL) is equal to:

000 Requires Read access rights

001 Requires Write access rights

010 Requires Read/Write access rights

Compatibility Mode

A file is considered to be in compatibility mode if the file is opened by:

- Any of the CREATE function calls
- An FCB function call
- A handle function call with compatibility mode specified.

A file can be opened any number of times in compatibility mode by a single process, provided that the file is not currently open under one of the other four sharing modes. If the file is marked read-only, and is open in DenyWrite sharing mode with Read Access, the file may be opened in Compatibility Mode with Read Access. If the file was successfully opened in one of the other sharing modes and an attempt is made to open the file again in Compatibility Mode, an interrupt 24H is generated to signal this error. The base interrupt 24H error indicates **Drive not ready**, and the extended error indicates a **Sharing violation**.

Sharing Modes

The sharing modes for a file opened in compatibility mode are changed by DOS 4.00 depending on the read-only attribute of the file. This allows sharing of read-only files.

File Opened By	Read-Only Access	Sharing Mode
FCB	Read-Only	DenyWrite
Handle Read	Read-Only	DenyWrite
Handle Write	Error	-----
Handle Read/Write	Error	-----

File Opened By	Not Read-Only Access	Sharing Mode
FCB	Read/Write	Compatibility
Handle Read	Read	Compatibility
Handle Write	Write	Compatibility
Handle Read/Write	Read/Write	Compatibility

DenyRead/Write Mode (Exclusive)

If a file is successfully opened in DenyRead/Write mode, access to the file is exclusive. A file currently open in this mode cannot be opened again in any sharing mode by any process (including the current process) until the file is closed.

DenyWrite Mode

A file successfully opened in DenyWrite sharing mode, prevents any other write access opens to the file (A = 001 or 010) until the file is closed. An attempt to open a file in DenyWrite mode is unsuccessful if the file is open with a write access.

DenyRead Mode

A file successfully opened in DenyRead sharing mode, prevents any other read sharing access opens to the file (A = 000 or 010) until the file is closed. An attempt to open a file in DenyRead sharing mode is unsuccessful if the file is open in Compatibility mode or with a read access.

DenyNone Mode

A file successfully opened in DenyNone mode, places no restrictions on the read/write accessibility of the file. An attempt to open a file in DenyNone mode is unsuccessful if the file is open in Compatibility mode.

When accessing files that reside on a network disk, no local buffering is done when files are opened in any of the following sharing modes:

- DenyRead
- DenyNone.

Therefore, in a network environment, DenyRead/Write sharing mode, Compatibility sharing mode, and DenyWrite mode opens are buffered locally.

The following sharing matrix shows the results of opening, and subsequently attempting to reopen the same file using all combinations of access and sharing modes:

3DH —
Open a File

		DRW			DW			DR			ALL		
		I	IO	O	I	IO	O	I	IO	O	I	IO	O
DRW	I	N	N	N	N	N	N	N	N	N	N	N	N
	IO	N	N	N	N	N	N	N	N	N	N	N	N
	O	N	N	N	N	N	N	N	N	N	N	N	N
DW	I	N	N	N	Y	N	N	N	N	N	Y	N	N
	IO	N	N	N	N	N	N	N	N	N	Y	N	N
	O	N	N	N	N	N	N	Y	N	N	Y	N	N
DR	I	N	N	N	N	N	N	N	N	N	N	N	Y
	IO	N	N	N	N	N	N	N	N	N	N	N	Y
	O	N	N	N	N	N	N	N	N	Y	N	N	Y
ALL	I	N	N	N	Y	Y	Y	N	N	N	Y	Y	Y
	IO	N	N	N	N	N	N	N	N	N	Y	Y	Y
	O	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y

Y :2nd,3rd,...open is allowed
N :2nd,3rd,...open is denied
DRW :DenyRead/Write Mode (Exclusive)
DW :DenyWrite Mode
DR :DenyRead Mode
ALL :Read/Write Mode
I :Read Only Access
O :Write Only Access
IO :Read/Write Access

Purpose

Closes the specified file handle.

Example

```
        MOV     AH,3EH          ; Function Call -
                                ; Close a Handle
        MOV     BX,Handle
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

----

Handle          DW      ?       ; File Handle (from Open / Create)
```

Comments

On entry, BX contains the file handle that was returned by Open or Create. On return, the file is closed, the directory is updated, and all internal buffers for that file are flushed.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

3FH —
Read from a File or Device

Purpose

Transfers the specified number of bytes from a file into a buffer location.

Example

```
MOV  AX,SEG Buffer              ; Address data buffer
MOV  DS,AX
MOV  DX,OFFSET Buffer
MOV  AH,3FH                     ; Function - Read from a file
MOV  BX,Handle
MOV  CX,BufSize                 ; Buffer size
INT  21H                        ; Issue request to DOS
JC   Error                      ; Error code in AX
CMP  AX,0                       ; At End Of File?
JE   EOF                        ; Yes!
MOV  SizeRead,AX                ; Save Amount Read

 ----
 N             EQU   512        ; Typical buffer size
 Handle        DW    ?          ; File Handle (from Open /Create)
 BufSize       DW    N          ; Buffer Size, N is
 Buffer        DB    N DUP(?)   ; Data Buffer
 SizeRead      DW    ?          ; Amount of Data in Buffer
```

Comments

On entry, BX contains the file handle. CX contains the number of bytes to read. DS:DX contains the buffer address. On return, AX contains the number of bytes read.

This function call attempts to transfer (CX) bytes from a file into a buffer location. It is not guaranteed that all bytes will be read. For example, when DOS 4.00 reads from the keyboard, at most one line of text is transferred. If this read is performed from the standard input device, the input can be redirected. If the value in AX is 0, then the program has tried to read from the end of file.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Network Access Rights: Requires Read access rights.

Purpose

Transfers the specified number of bytes from a buffer into a specified file.

Example

```
MOV  AX,SEG Buffer       ;Data Buffer
MOV  DS,AX
MOV  DX,OFFSET Buffer
MOV  CX,BufSize
MOV  AH,40H              ;Function-Write to a File
MOV  BX,Handle
MOV  DX,OFFSET Buffer
INT  21H                 ; Issue request to DOS
JC   Error               ; Error code in AX
CMP  AX,CX               ; Disk Full?
JB   FullDisk            ; Yes!

 ----
 N            EQU   512      ; Typical buffer size
 Handle       DW    ?        ; File Handle (from Open / Create)
 BufSize      DW    N        ; Buffer Size
 Buffer       DB    N DUP(?) ; Data Buffer
```

Comments

On entry, BX contains the file handle. CX contains the number of bytes to write. DS:DX contains the address of the data to write.

This function call attempts to transfer (CX) bytes from a buffer into a file. AX returns the number of bytes actually written. If the carry flag is not set and this value is not the same as the number requested (in CX), it should be considered an error. Although no error code is returned, your program can compare these values. Normally, the reason for the error is a full disk. If this write is performed to the standard output device, the output can be redirected.

To truncate a file at the current position of the file pointer, set the number of bytes (CX) to 0 before issuing the interrupt 21H. The file pointer can be moved to the desired position by reading, writing, and performing function call 42H, (Move File Read/Write Pointer.)

40H —
Write to a File or Device

If the file is read-only, the write to the file or device is not performed.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Network Access Rights: Requires Write access rights.

41H — Delete a File from a Specified Directory (UNLINK)

Purpose

Removes a directory entry associated with a filename.

Example

```
        MOV  AX,SEG FName        ; File Name
        MOV  DS,AX
        MOV  DX,OFFSET FName
        MOV  AH,41H              ; Function-Delete a File
        INT  21H                 ; Issue request to DOS
        JC   Error               ; Error code in AX

  ----

FName   DB    "?? .. ??",0  ; ASCIIZ Name
                            ;   example: "c:\dir\File.ext",0
```

Comments

Global filename characters are not allowed in any part of the ASCIIZ string. Read-only files cannot be deleted by this call. To delete a read-only file, you can first use call 43H to change the file's read-only attribute to 0, then delete the file.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Network Access Rights: Requires Create access rights.

42H —
Move File Read Write Pointer (LSEEK)

Purpose
Moves the read/write pointer according to the method specified.

Example

```
        MOV     AH,42H                  ; Function Call -
                                        ; Move Read/Write Pointer
        MOV     AL,Method               ; Method of Positioning:
                                        ;   0 = From Beginning of File
                                        ;       (BOF)
                                        ;   1 = From Current Position
                                        ;   2 = From End of File (EOF)
        MOV     BX,Handle               ; Select File
        MOV     DX,WORD PTR Position+0  ; New Position = Position + METHOD
        MOV     CX,WORD PTR Position+2
        INT     21H                     ; Issue request to DOS
        JC      Error                   ; Error code in AX
        MOV     WORD PTR Position+0,AX  ; Set new File Position
        MOV     WORD PTR Position+2,DX

----

Handle          DW      ?               ; File Handle (from Open /
                                        ; Create)
Position        DD      ?               ; File Offset (may be
                                        ; negative)
Method          DB      ?
```

Comments
On entry, AL contains a method value. BX contains the file handle. CX:DX contains the desired offset in bytes with CX containing the most significant part. On return, DX:AX contains the new location of the pointer with DX containing the most significant part if the carry flag is not set.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

42H —
Move File Read Write Pointer (LSEEK)

This function call moves the read/write pointer according to the following methods:

AL	Description
0	The pointer is moved CX:DX bytes (offset) from the beginning of the file.
1	The pointer is moved to the current location plus offset.
2	The pointer is moved to the end-of-file plus offset. This method can be used to determine file's size.

Note: If an LSEEK operation is performed on a file that resides on a network disk that is open in either DenyRead or DenyNone sharing mode, the read/write pointer information is adjusted on the computer where the file actually exists. If the file is opened in any other sharing mode, the read/write pointer information is kept on the remote computer.

43H —
Change File Mode (CHMOD)

Purpose
Changes the file mode of the specified file.

Example

```
; To Get Attributes

        MOV     AX,SEG FName            ;File Name
        MOV     DS,AX
        MOV     DX,OFFSET FName         ;DS:DX points to ASCIIZ path name
        MOV     AL,0                    ;Indicate get
        MOV     AH,43H                  ;Function-Change File Mode
        INT     21H                     ;Issue request to DOS
        JC      Error                   ;Error code in AX
        MOV     Attribute,CX            ;Save Attribute

; To Set Attributes

        MOV     AX,SEG FName            ;File Name
        MOV     DS,AX
        MOV     DX,OFFSET FName         ;DS:DX points to ASCIIZ path name
        MOV     AL,1                    ;Indicate set
        MOV     AH,43H                  ;Function-Change File Mode
        MOV     CX,Attribute            ;Set Attribute
        INT     21H                     ;Issue request to DOS
        JC      Error                   ;Error code in AX

----


Fname           DB      64 Dup (0)      ;ASCIIZ Name
                                        ;  example: "c:\dir\File.ext",0
Attribute       DW      ?               ;File Attribute
                                        ;  example: 0001H to set Read-Only
```

Comments
On entry, AL contains a function code, and DS:DX contains the address of an ASCIIZ string with the drive, path, and filename.

If AL contains 01H, the file's attribute is set to the attribute in CX. See "The Disk Directory" on page 2-3 for the attribute byte description. If AL is 00H the file's current attribute is returned in CX.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Note: Only the Archive (20H), Read-Only (01H), System (04H) and Hidden (02H) bits can be changed. All other bits of CX must be 0, otherwise, an error may be indicated.

Network Access Rights: To change the archive bit (AL=20H), no access rights are required. To change any other bit, Create access rights are required.

44H —
I/O Control for Devices

Purpose

Sets or gets device information associated with open device handles, or sends control strings to the device handle or receives control strings from the device handle.

See Appendix C for full details on this function call.

Purpose

Returns a new file handle for an open file that refers to the same file at the same position.

Example

```
        MOV     AH,45H          ; Function Call -
                                ; Duplicate a Handle
        MOV     BX,Handle       ; Select File
        INT     21H             ; Issue request to the operating system
        JC      Error           ; Error code in AX
        MOV     NewHandle,AX    ; Save New Handle

----

Handle      DW      ?       ; File Handle (from Open / Create)
NewHandle   DW      ?       ; File Handle that duplicates Handle
```

Comments

On entry, BX contains the file handle. On return, AX contains the returned file handle.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Note: If you move the read/write pointer of either handle by a read, write, or LSEEK function call, the pointer for the other handle is also changed.

46H —
Force a Duplicate of a Handle (FORCDUP)

Purpose

Forces the handle in CX to refer to the same file at the same position as the handle in BX.

Example

```
        MOV     AH,46H          ; Function Call -
                                ; Force Duplicate a Handle
        MOV     BX,Handle       ; Select file
        MOV     CX,NewHandle    ; Select new definition of File
        INT     21H             ; Issue request to the operating system
        JC      Error           ; Error code in AX

----

Handle          DW      ?       ; File Handle (from Open/Create)
NewHandle       DW      ?       ; File Handle that duplicates Handle
```

Comments

On entry, BX contains the file handle. CX contains a second file handle. On return, the CX file handle refers to the same file at the same position as the BX file handle. If the CX file handle was an open file, it is closed first. If you move the read/write pointer of either handle, the pointer for the other handle is also changed.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Purpose

Places the full path name (starting from the root directory) of the current directory for the specified drive in the area pointed to by DS:SI.

Example

```
MOV     AX,SEG DName     ; Directory Name Buffer
MOV     DS,AX
MOV     SI,OFFSET DName  ; OS:SI points to buffer
MOV     DL,Drive         ; Select Drive
MOV     AH,47H           ; Function-Get Current Dir
INT     21H              ; Issue request to the operating system
JC      Error            ; Error code in AX

----

Drive   DB  ?            ; Drive (0=current, 1=A:, 2=b:, ...)
DName   DB  64 DUP(?)    ; ASCIIZ Directory Name Returned
                         ;   example: "dir1\dir2",0
```

Comments

The drive letter is not part of the returned string. The string does not begin with a backslash and is terminated by a byte containing 00H.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

48H —
Allocate Memory

Allocates the requested number of paragraphs of memory.

Example

```
        MOV     AH,48H          ; Function Call -
                                ; Allocate Memory
        MOV     BX,Paragraphs   ; Paragraphs Desired
        INT     21H             ; Issue request to the operating system
        JNC     Done
        MOV     AH,48H          ; Function Call -
                                ; Allocate memory
                                ; BX set to largest available memory
        INT     21H             ; Issue request to the operating system
Done:
        MOV     BlockSeg,AX     ; Save BlockSeg of memory
        MOV     Paragraphs,BX

----

Paragraphs      DW      ?       ; Size requested in paragraphs
                                ; (Bytes allocated is 16 * Paragraphs)
BlockSeg        DW      ?       ; BlockSeg address of allocated memory
```

Comments

On entry, BX contains the number of paragraphs requested. On return, AX:0 points to the allocated memory block. If the allocation fails, BX returns the size of the largest block of memory available in paragraphs.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Purpose

Frees the specified allocated memory.

Example

```
        MOV     AH,49H          ; Function Call - Free Memory
        MOV     ES,BlockSeg     ; Set address to free
        INT     21H             ; Issue request to the operating system
        JC      Error

  ----

  BlockSeg      DW      ?       ; BlockSeg address of allocated memory
```

Comments

On entry, ES contains the segment of the block to be returned to the system pool. On return, the block of memory is returned to the system pool.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

4AH —
Modify Allocated Memory Blocks (SETBLOCK)

Purpose

Modifies allocated memory blocks to contain the new specified block size.

Example

```
        MOV     AH,4AH          ; Function Call -
                                ; Modify Allocated Memory; allocate memory
        MOV     ES,BlockSeg     ; Set address to free
        MOV     BX,BlockSize    ; New size (may be larger or smaller)
        INT     21H             ; Issue request to the operating system
        JNC     Done
        MOV     AH,4AH          ; Function Call -
                                ; Allocate memory
                                ; BX set to largest available Size
        INT     21H             ; Issue request to the operating system
Done:
        MOV     Size,BX

----

BlockSeg        DW      ?       ; Segment address of allocated memory
BlockSize       DW      ?       ; Size requested in paragraphs
                                ; (Bytes allocated is 16 * Size)
```

Comments

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

Note: This call is often used to set the size of a program before using function call 31H (Terminate Process and Remain Resident). Use the program segment prefix. This value can be obtained using function call 62H (Get Program Segment Prefix Address). Another use is to release memory to prepare for using function call 4BH (Load or Execute a Program).

Purpose

Allows a program to load another program into memory and may choose to begin execution of it.

Example

```
; To Execute a Program

 MOV  AH,4BH                  ; Function Call – Execute a Program
 MOV  AL,0                    ; Indicate execute program
 MOV  CX,SEG Parms            ; Program parameters
 MOV  ES,CX
 MOV  BX,OFFSET Parms         : ES:BX points to parameter block
 MOV  CX,SEG PName            ; Program name
 MOV  DS,CX
 MOV  DX,OFFSET PName         ; DS:DX points to program name
 MOV  WORD PTR StackSave+0,SP ; Save stack pointer
 MOV  WORD PTR StackSave+2,SS
 INT  21H                     ; Issue request to DOS
 JC   Error                   ; Error code in AX
; Note:  All Registers (except CS:IP) Destroyed

; Program Runs here

 CLI                          ; Protect from stack usage
 MOV  SS,WORD PTR StackSave+2 ; Restore stack pointer
 MOV  SP,WORD PTR StackSave+0
 STI                          ; Enable interrupts
 MOV  AH,4DH                  ; Function Call – Get Return Code
 INT  21H                     ; Issue request to DOS
 MOV  RetCode,AX              ; Save return code

; To Load an Overlay
 MOV  AH,4BH                  ; Function Call – Execute a Program
 MOV  AL,3                    ; Indicate load overlay
 MOV  CX,SEG OParms           ; Overlay parameters
 MOV  ES,CX
 MOV  BX,OFFSET OParms        ; ES:BX points to parameter block
 MOV  CX,SEG PName            ; Overlay name
 MOV  DS,CX
 MOV  DX,OFFSET PName         ; DS:DX points to overlay filename
 INT  21H                     ; Issue request to DOS
 JC   Error                   ; Error code in AX
```

4BH —
Load or Execute a Program (EXEC)

```
----
PName       DB    64 Dup (0)   ; ASCIIZ Name
                                ;   example: "c:\dir\File.ext",0
Parms       LABEL WORD         ; Program parameters
Env@        DW    ?            ; Environment segment address
                                ;   Value of 0000H indicates copy EXEC'ers
                                ;   Environment
Cmd@        DD    ?            ; Command line address
FCB1@       DD    ?            ; FCB Image to set to New PSP+5CH
FCB2@       DD    ?            ; FCB Image to set to New PSP+6CH

StackSave   DD    ?            ; Stack pointer save area
RetCode     DW    ?            ; Program return code
                                ; (see function code 4DH for more information)
OParms      LABEL WORD         ; Overlay parameters
Load@       DW    ?            ; Overlay load segment address
RelocFactor DW    ?            ; Relocation factor to apply (for .EXE files)
```

Comments

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

The following function values are allowed in AL:

Function Value	Description
00H	Load and execute the program. A program segment prefix is established for the program; terminate and Ctrl-Break addresses are set to the instruction after the EXEC system call. **Note:** When control is returned, all registers are changed, including the stack. You must restore SS, SP, and any other required registers before proceeding.
03H	Load, do not create the program segment prefix, and do not begin execution. This is useful in loading program overlays.

All open files of a process are duplicated in the newly created process after an EXEC, unless the file was opened with the inheritance bit set to 1. This means that the parent process has control over the meanings of standard input, output, auxiliary, and printer devices. The parent could, for example, write a series of records to a file, open the file as standard input, open a listing file as standard output, and then execute a sort program that takes its input from standard input and writes to standard output.

Also inherited (or copied from the parent) is an "environment." This is a block of text strings (less than 32KB total) that conveys various

configuration parameters. The following is the format of the environment (always on a paragraph boundary):

Byte ASCIIZ string 1
Byte ASCIIZ string 2
...
Byte ASCIIZ string n
Byte of 0

Typically the environment strings have the form:

parameter=value

Following the byte of 0 in the environment is a WORD that indicates the number of other strings following. Following this is a copy of the DS:DX filename passed to the child process. For example, the string VERIFY=ON could be passed. A 0 value of the environment address causes the newly created process to inherit the original environment unchanged. The segment address of the environment is placed at offset 2CH of the program segment prefix for the program being invoked.

Errors codes are returned in AX. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned.

Note: When your program received control, all available memory was allocated to it. You must free some memory (see call 4AH) before EXEC can load the program you are invoking. Normally, you would shrink down to the minimum amount of memory you need, and free the rest.

Purpose

Terminates the current process and transfers control to the invoking process.

Example

```
        MOV     AH,4CH          ; Function Call - Terminate a Process
        MOV     AL,ErrorCode    ; Set ERRORLEVEL
        INT     21H             ; Issue request to DOS
        INT     20H             ; Be safe if running on PC/DOS 1.1

----

ErrorCode       DB      ?       ; Error Code (sets ERRORLEVEL if EXEC'ed
                                ; by COMMAND.COM)
```

Comments

In addition, a return code can be sent. The return code can be interrogated by the batch subcommands IF and ERRORLEVEL and by the wait function call 4DH. All files opened by this process are closed.

4DH —
Get Return Code of a Subprocess (WAIT)

Purpose

Gets the return code specified by another process either through function call 4CH or function call 31H. It returns the Exit code only once.

Example

```
        MOV     AH,4DH          ; Function Call - Get Return Code
        INT     21H             ; Issue request to DOS
        MOV     RetCode,AX      ; Save return code

RetCode         LABEL   WORD    ; Program return code
ExitCode        DB      ?       ; ERRORLEVEL value
ExitType        DB      ?       ; Method used to exit:
                                ;   00H - for normal termination
                                ;   01H - for termination by Ctrl-Break
                                ;   02H - for termination as a result
                                ;         of a critical device error
                                ;   03H - for termination by call 31H
```

Comments

The low byte of the exit code contains the information sent by the exiting routine.

Purpose

Finds the first filename that matches the specified file specification.

Example

```
        MOV     AH,1AH          ; Function Call - Set DTA Address
        MOV     CX,SEG DTA      ; Address buffer for found file
        MOV     DS,CX
        MOV     DX,OFFSET DTA
        INT     21H             ; Issue request to DOS
        MOV     AH,4EH          ; Function Call - ASCIIZ Find First
        MOV     CX,SEG FName    ; Directory or filename
        MOV     DS,CX
        MOV     DX,OFFSET FName ; DS:DX points to ASCII filename
        MOV     CX,Attribute    ; Set Match Attribute
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX


----

DTA             LABEL   BYTE            ; Find return information
                DB      21 DUP(0)       ; Reserved for DOS 4.00 to continue find
FileAttr        DB      ?               ; Matched files attribute low byte
FileTime        DW      ?               ; File time
FileDate        DW      ?               ; File date
FileSize        DD      ?               ; File size
FileNameExt     DB      "????????.???",0 ; Filename and extension

FName           DB      64 DUP (0)      ; ASCIIZ Name
                                        ;   example: "c:\dir\*.*",0
Attribute       DW      ?               ; Select files attribute
                                        ; Combination of following:
                                        ;  0002H=Hidden
                                        ;  0004H=System
                                        ;  0008H=Volume label
                                        ;  0010H=Directory
```

Notes:

If an extended FCB is used, the following search pattern is used:

1. If the attribute is 0, only normal file entries are found. Entries for volume label, sub-directories, hidden and system files are not returned.
2. If the attribute field is set for hidden or system files, or directory entries, it is to be considered as an inclusive search. All normal

file entries plus all entries matching the specified attributes are returned. To look at all directory entries except the volume label, the attribute byte may be set to hidden + system + directory (all 3 bits on).

3. If the attribute field is set for the volume label, it is considered an exclusive search, and *only* the volume label entry is returned.

Comments

The filename in DS:DX can contain global filename characters. The ASCIIZ string cannot contain a network path. See function call 11H (Search for First Entry) for a description of how the attribute bits are used for searches.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

Note: The name and extension of file found is returned as an ASCIIZ string. All blanks are removed from the name and extension, and, if an extension is present, it is preceded by a period.

4FH — Find Next Matching File (FIND NEXT)

Purpose

Finds the next directory entry matching the name that was specified on the previous Find First or Find Next function call.

Example

```
        MOV     AH,1AH          ; Function Call - Set DTA Address
        MOV     CX,SEG DTA      ; Address buffer for found file
        MOV     DS,CX
        MOV     DX,OFFSET DTA
        INT     21H             ; Issue request to DOS
        MOV     AH,4FH          ; Function Call - Find next
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX


----

DTA             LABEL   BYTE            "  ; Find Return Information
                DB      21 DUP(0)       "  ; Reserved for DOS to Continue Find
                                        "  ; Set by Find First or Previous Find Next
FileAttr        DB      ?               "  ; Matched Files Attribute Low Byte
FileTime        DW      ?               "  ; File Time
FileDate        DW      ?               "  ; File Date
FileSize        DD      ?               "  ; File Size
FileNameExt     DB      "????????.???",0 ; File Name and Extension
```

Comments

If a matching file is found, the DTA is set as described in call 4EH (Find First Matching File (FIND FIRST)). If no more matching files are found, an error code is returned.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location.

54H —
Get Verify Setting

Purpose
Returns the value of the verify flag.

Example

```
        MOV     AH,54H          ; Function Call - Get VERIFY Setting
        INT     21H             ; Issue request to DOS
        MOV     VERIFY,AL       ; Save VERIFY State
  -----

VERIFY          DB      ?       ; VERIFY State:
                                ; 0 = OFF
                                ; 1 = ON
```

Comments
On return, AL returns 00H if verify is OFF, 01H if verify is ON. Note that the verify switch can be set through call 2EH (Set/Reset Verify Switch).

Purpose

Renames the specified file.

Example

```
        MOV     AH,56H              ; Function Call - ASCIIZ Rename File
        MOV     CX,SEG FName        ; File Name
        MOV     DS,CX
        MOV     DX,OFFSET FName     ; DS:DX points to original name
        MOV     CX,SEG NewName      ; New File Name
        MOV     ES,CX
        MOV     DI,OFFSET NewName   ; ES:DI points to rename
        INT     21H                 ; Issue request to DOS
        JC      Error               ; Error code in AX

----

FName           DB      64 DUP (0)    ; ASCIIZ Name
                                      ;   example: "c:\dir\abc.lst",0
NewName         DB      64 DUP (0)    ; ASCIIZ Name
                                      ;   example: "\dir\xyz.lst",0
```

Comments

If a drive is used in the NewName string, it must be the same as the drive specified or implied in the Name string. The directory paths need not be the same, allowing a file to be moved to another directory and renamed in the process. Directory names can be changed but not moved. Global filename characters are not allowed in the filename.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

Network Access Rights: Requires Create access rights.

57H —
Get/Set File's Date and Time

Purpose
Gets or sets a file's date and time.

Example
```
; To Get a File's Date and Time

        MOV     AH,57H          ; Function Call - Get/Set Date and Time
        MOV     AL,0            ; Indicate get
        MOV     BX,Handle       ; Select file
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     FileTime,CX     ; Save Time
        MOV     FileDate,DX     ; Save Date

; To Set a File's Date and Time

        MOV     AH,57H          ; Function Call - Get/Set Date and Time
        MOV     AL,1            ; Indicate Set
        MOV     BX,Handle       ; Select file+
        MOV     CX,FileTime     ; Set Time
        MOV     DX,FileDate     ; Set Date
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

----

Handle          DW      ?       ; File Handle (from Open / Create)
FileTime        DW      ?       ; File Time
FileDate        DW      ?       ; File Date
```

Comments
The date and time formats are the same as those for the directory entry described in Chapter 5 of this book.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

Purpose

Returns additional error information, such as the error class, location, and recommended action.

Example

```
        PUSH    DX                  ; Save Registers
        PUSH    SI
        PUSH    DI
        PUSH    ES
        PUSH    DS
        MOV     AH,59H              ; Function Call - Get Extended Error
        MOV     BX,0                ; Version 0 information
        INT     21H                 ; Issue request to DOS
        POP     DS                  ; Restore registers
        POP     ES
        POP     DI
        POP     SI
        POP     DX
        MOV     ExtError,AX         ; Save error code
        MOV     ErrorClass,BH       ; Save error class
        MOV     ErrorAction,BL      ; Save error action
        MOV     ErrorLocation,CH    ; Save error location

----

ExtError        DW      ?           ; DOS extended error
ErrorClass      DB      ?           ; Class of error
ErrorAction     DB      ?           ; Suggested action
ErrorLocation   DB      ?           ; System area effected
```

Comments

This function call returns the error class, location, and recommended action, in addition to the return code. Use this function call from:

- Interrupt 24H error handlers
- Interrupt 21H function calls that return an error in the carry bit
- FCB function calls that return FFH.

On return, the registers contents of DX, SI, DI, ES, CL, and DS are destroyed.

59H —
Get Extended Error

Error Return In Carry Bit

For function calls that indicate an error by setting the carry flag, the correct method for performing function call 59H is:

- Load registers.
- Issue interrupt 21H.
- Continue operation, if carry not set.
- Disregard the error code and issue function call 59H to obtain additional information.
- Use the value in BL to determine the suggested action to take.

Error Status In AL

For function calls that indicate an error by setting AL to FFH, the correct method for performing function call 59H is:

- Load registers.
- Issue interrupt 21H.
- Continue operation, if error is not reported in AL.
- Disregard the error code and issue function call 59H to obtain additional information.
- Use the value in BL to determine the suggested action to take.

Purpose

Generates a unique filename, and creates that file in the specified directory.

Example

```
        MOV     AH,5AH                  ; Function Call - Create a
                                        ; Unique File
        MOV     CX,SEG DirName          ; Directory name
        MOV     DS,CX
        MOV     DX,OFFSET DirName
        MOV     CX,Attribute            ; File attribute
        INT     21H                     ; Issue request to DOS
        JC      Error                   ; Error code in AX
        MOV     Handle,AX               ; Save file handle for following
                                        ; Operations

----

DirName         DB      "?? .. ??\",0   ; ASCIIZ name
                DB      "????????.???"
                                        ;   example in : "c:\dir\",0
                                        ;   example out: "c:\dir\file",0
Handle          DW      ?               ; File handle
Attribute       DW      ?               ; Select file's attribute
```

Comments

On entry, AH contains 5AH. If no error has occurred, the file is opened in compatibility mode with Read/Write access. The read/write pointer is set at the first byte of the file and AX contains the file handle and the filename is appended to the path specified in DS:DX.

This function call generates a unique name and attempts to create a new file in the specified directory. If the file already exists in the directory, then another unique name is generated and the process is repeated. Programs that need temporary files should use this function call to generate unique filenames.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6

and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

Note: The file created using this function call is not automatically deleted at program termination.

Network Access Rights: Requires Create access rights.

Purpose

Creates a new file.

Example

```
        MOV     AH,5BH          ; Function Call - Create a New File
        MOV     CX,SEG FName    ; File Name
        MOV     DS,CX
        MOV     DX,OFFSET FName
        MOV     CX,Attribute
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     Handle,AX       ; Save File Handle for following operations

----

FName           DB      64 DUP (0)      ; ASCIIZ Name
                                        ;   example: "c:\dir\file",0
Handle          DW      ?               ; File Handle
Attribute       DW      ?               ; Select File's Attribute
```

Comments

This function call is the same as function call 3CH (Create), except it will fail if the filename already exists. The file is created in compatibility mode for reading and writing and the read/write pointer is set at the first byte of the file.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

Network Access Rights: Requires Create access rights.

5CH —
Lock/Unlock File Access

Purpose

Locks or unlocks a single range of bytes in an opened file. This function call provides database services that are useful in maintaining database integrity in a network environment.

Example

```
; To Lock a Single Range

      MOV     AH,5CH                  ; Function Call -
                                      ; Access Lock/Unlock File
      MOV     AL,0                    ; Indicate lock
      MOV     BX,Handle               ; Select file
      MOV     DX,WORD PTR Position+0  ; Set position
      MOV     CX,WORD PTR Position+2
      MOV     DI,WORD PTR Llength+0   ; Set length
      MOV     SI,WORD PTR Llength+2
      INT     21H                     ; Issue request to DOS
      JC      Error                   ; Error code in AX

; To Unlock a Single Range

      MOV     AH,5CH                  ; Function Call -
                                      ; Lock/Unlock File Access
      MOV     AL,1                    ; Indicate unlock
      MOV     BX,Handle               ; Select file
      MOV     DX,WORD PTR Position+0  ; Set position
      MOV     CX,WORD PTR Position+2
      MOV     DI,WORD PTR Llength+0   ; Set length
      MOV     SI,WORD PTR Llength+2
      INT     21H                     ; Issue request to DOS
      JC      Error                   ; Error code in AX

----

Handle        DW      ?             ; File Handle
Position      DD      ?             ; Start of Range
Llength       DD      ?             ; Length of Range
```

Comments

The Lock/Unlock function calls should only be used when a file is opened using the DenyRead or DenyNone sharing modes. These modes do no local buffering of data when accessing files on a network disk.

AL = 00H Lock

Provides a simple mechanism for excluding other processes' read/write access to regions of the file. If another process attempts to read or write in such a region, its system call is retried the number of times specified with the system retry count set by IOCTL. If, after those retries, no success occurs, a general failure error is generated, signaling the condition. The number of retries, as well as the length of time between retries, can be changed using function call 440BH (IOCTL Change Sharing Retry Count).

The recommended action is to issue function call 59H (Get Extended Error) to get the error code, in addition to the error class, location, and recommended action. The locked regions can be anywhere in the logical file. Locking beyond end-of-file is not an error. It is expected that the time in which regions are locked will be short. Duplicating the handle duplicates access to the locked regions. Access to the locked regions is not duplicated across the EXEC system call. Exiting with a file open and having issued locks on that file has undefined results.

Programs that may be cancelled using INT 23H (File Size) or INT 24H (Set Relative Record Field) should trap these and release the locks before exiting. The proper method for using locks is not to rely on being denied read or write access, but to attempt to lock the region desired and examining the error code.

AL = 01H Unlock

Unlock releases the lock issued in the lock system call. The region specified must be exactly the same as the region specified in the previous lock. Closing a file with locks still in force has undefined results. Exiting with a file open and having issued locks on that file has undefined results.

Programs that may be aborted using INT 23H (File Size) or INT 24H (Set Relative Record Field) should trap these and release the lock before exiting. The proper method for using locks is not to rely on being denied read or write access, but attempting to lock the region desired and examining the error code.

5CH —
Lock/Unlock File Access

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

Purpose

Returns the character identifier of the local computer.

Example

```
MOV     AX,SEG CNAME        ; Name buffer
MOV     DS,AX
MOV     DX OFFSET CNAME
MOV     AX,5E00H            ; Function Call -
                            ; Get Machine Name
INT     21H                 ; Issue request to DOS
JC      Error               ; Error code in AX
MOV     NameFlag,CH         ; Save name number indicator
MOV     NameID,CL           ; Save NETBIOS name number

  ----

CName           DB      "??????????????",0    ; ASCIIZ computer name
NameFlag        DB      ?                     ; 0 = Name is not set
                                              ; 1 = Name is set
NameID          DB      ?                     ; NETBIOS name number
```

Comments

Get Machine Name returns the text of the current computer name to the caller. The computer name is a 15-character byte string padded with spaces and followed by a 00H byte. If the computer name was never set, register CH is returned with 00H and the value in the CL register is invalid. The IBM PC Local Area Network program must be loaded for the function call to execute properly.

5E02H —
Set Printer Setup

Purpose
Specifies an initial string for printer files.

Example

```
        MOV     AX,5E02H        ; Function Call -
                                ; Set Printer Setup
        MOV     BX,Index        ; Redirection List Index
        MOV     CX,size         ; String size
        MOV     SI,SEG String   ; String Buffer
        MOV     DS,SI
        MOV     SI,OFFSET String
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

----

Index           DW      ?         ; Redirection List Index
Size            DW      N         ; String size (Maximum 64)
String          DB      N DUP(?)  ; Printer Setup String
```

Comments
The string specified is put in front of all files destined for a particular network printer. Set Printer Setup allows multiple users of a single printer to specify their own mode of operation for the printer. BX is set to the same index that is used in function call 5F02H (Get Redirection List Entry). An error code is returned if print redirection is paused or if the IBM PC Local Area Network program is not loaded.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

IMPORTANT: The redirection index value may change if function call 5F03H (Redirect Device) or function call 5F04H (Cancel Redirection) is issued between the time the redirection list is scanned and the function call 5E02H (Set Printer Setup) is issued. Therefore, we recommend that you issue Set Printer Setup immediately after you issue "Get Redirection List ."

Purpose

Returns the printer setup string for printer files.

Example

```
        MOV     AX,5E03H        ; Function Call -
                                ; Get Printer Setup
        MOV     BX,Index        ; Redirection List Index
        MOV     CX,SEG String   ; String Buffer
        MOV     ES,CX
        MOV     DI,OFFSET String
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     Ssize, CX       ; Save String size

----

Index           DW    ?         ; Redirection List Index
Ssize           DW    ?         ; String size
String          DB    64 DUP(?) ; Printer Setup String
```

Comments

This function call returns the printer setup string which was specified using the function call 5E02H (Set Printer Setup). The setup string is attached to all files destined for a particular printer. The value in BX is set to the same index issued in function call 5F02H (Get Redirection List). Error code 1 (invalid function number) is returned if the IBM PC Local Area Network is not loaded.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

IMPORTANT: The redirection index value may change if function call 5F03H (Redirect Device) or function call 5F04H (Cancel Redirection) is issued between the time the redirection list is scanned and the function call 5E03H (Get Printer Setup) is issued. Therefore, we recommend that you issue "Get Printer Setup" immediately after you issue "Get Redirection List."

5F02H —
Get Redirection List Entry

Purpose
Returns nonlocal network assignments.

Example

```
      MOV     BX,0                ; Start at beginning of list
Get_Loop:                         ; Get next entry
      MOV     Index,BX            ; Redirection list index
      MOV     AX,5F02H            ; Function Call -
                                  ; Get redirection list entry
      MOV     SI,SEG device       ; "PRN" is possible
      MOV     DS,SI
      MOV     SI,OFFSET device    ; DS:SI points to local name
      MOV     DI,SEG info
      MOV     ES,DI
      MOV     DI,OFFSET inf       ; ES:DI points to buffer address of network name
      PUSH    BX
      PUSH    DX                  ; Save registers
      PUSH    BP
      INT     21H                 ; Issue request to DOS
      POP     BP                  ; Restore registers
      POP     DX
      JC      CheckEnd            ; Error code in AX
      MOV     Status,BH           ; Save status
      MOV     Type,BL             ; Save type
      MOV     UserParm,CX         ; Save user parmameter
      POP     BX
      INC     BX                  ; Set to next entry
      JMP     Get_Loop
```

```
CheckEnd:
      POP     BX                ; Balance state
      CMP     AX,18             ; End of list?
      JNE     Error             ; No!

----

Index       DW      ?             ; Redirection list index (0 based)
Device      DB      128 DUP(?)    ; ASCIIZ device name
                                  ;  example: "LPT1",0
                                  ;               "A:",0
Status      DB      ?             ; Device status
                                  ; Bit 0=0 : Device is OK
                                  ; Bit 0=1 : Device in Error
                                  ; Bit 7-1 reserved
UserParm    DW      ?             ; User parameter
Type        DB      ?             ; Device type
                                  ; 3 = NET USE device
                                  ; 4 = NET USE drive
Info        DB      128 DUP(?)    ; NET USE network path
                                  ;   example: "\\MYNODE\CDRIVE",0
```

Comments

The Get Redirection List Entry function call returns the list of network redirections that were created through function call 5F03H (Redirect Device). Each call returns one redirection, so BX should be increased by one each time to step through the list. The contents of the list may change between calls. The end-of-list is detected by error code 18 (no more files). Error code 1 (invalid function number) is returned if the IBM PC Local Area Network program is not loaded.

If either disk or print redirection is paused, the function is not effected.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

5F03H —
Redirect Device

Purpose
Causes a Redirector/Server connection to be made.

Example

```
        MOV     AX,5F03H                ; Function Call -
                                        ; Redirect Device
        MOV     SI,SEG Device           ; Device Buffer
        MOV     DS,SI
        MOV     SI,OFFSET Device
        MOV     DI,SEG Net Path info    ; Information Buffer
        MOV     ES,DI
        MOV     DI,OFFSET Net Path
        MOV     BL,Type                 ; Set Type
        MOV     CX,UserParm             ; Set User Parameter
        INT     21H                     ; Issue request to DOS
        JC      Error                   ; Error code in AX

----

Device          DB      "....",0        ; ASCIIZ Device Name
                                        ;  example: "LPT1",0
                                        ;               "A:",0
UserParm        DW      ?               ; User Parameter
Type            DB      ?               ; Device Type
                                        ; 3 = NET USE Device
                                        ; 4 = NET USE Drive
Net Path        DB      128 DUP(0)
```

Comments
This call defines the current directories for the network and defines redirection of network printers.

- If BL = 03, the source specifies a printer, the destination specifies a network path, and the CX register has a word that DOS 4.00 maintains for the programmer. For compatibility with the IBM PC Local Area Network program, CX should be set to 0. Values other than 0 are reserved for the IBM PC Local Area Network program. This word may be retrieved through function call 5F02H (Get Redirection List). All output destined for the specified printer is buffered and sent to the remote printer spool for that device. The printers are redirected at the INT 17H level.

The source string must be **PRN** , **LPT1**, **LPT2**, or **LPT3** each ended with a 00H. The destination string must point to a network name string of the following form:

[*computername*\{*shortname*|*printdevice*}]

The destination string must be ended with a 00H.

The ASCIIZ password (0 to 8 characters) for access to the remote device should immediately follow the network string. The password must end with a 00H. A null (0 length) password is considered to be no password.

- If BL = 4, the source specifies a drive letter and colon ending with 00H, the destination specifies a network path ending with 00H, and the CX register has a word that DOS maintains for the programmer. For compatibility with the IBM PC Local Area Network program, CX should be set to 00H. Values other than 00H are reserved for the IBM PC Local Area Network program. The value may be retrieved through function call 5F02H (Get Redirection List). If the source was a drive letter, the association is made between the drive letter and the network path. All subsequent references to the drive letter are translated to references to the network path. If the source is an empty string, the system attempts to grant access to the destination with the specified password without redirecting any device.

  The ASCIIZ password for access to the remote path should immediately follow the network string. A null (0 length) password ended with 00H is considered to be no password.

  Error codes are returned in AX. Issue function call 59H for additional information about the error class, suggested action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H (Get Extended Error) .

5F03H —
Redirect Device

Notes:

1. Devices redirected through this function call are not displayed by the NET USE command.
2. An error is returned if you try to redirect a drive while disk redirection is paused, or if you try to redirect a printer while print redirection is paused.

Purpose

Cancels a previous redirection.

Example

```
        MOV     AX,5F04H        ; Function Call -
                                ; Cancel redirection
        MOV     SI,SEG Device   ; Device buffer
        MOV     DS,SI
        MOV     SI,OFFSET Device
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

----

Device          DB      "....",0        ; ASCIIZ Device Name
                                        ;  example: "LPT1",0
                                        ;           "A:",0
                                        ;           "\\Computer\Path",0
```

Comments

The redirection created by the Redirect Device function call (5F03H) is removed through the Cancel Redirection call. If the buffer points to a drive letter and the drive is associated with a network name, the association is ended and the drive is restored to its physical meaning. If the buffer points to PRN, LPT1, LPT2, or LPT3, and the device has an association with a network device, the association is terminated and the device is restored to its physical meaning. If the buffer points to a network path ending with 00H and a password ending with 00H, the association between the local machine and the network directory is terminated.

An error is returned if you try to cancel a redirected file device while disk redirection is paused, or if you try to cancel a redirected printer while print redirection is paused. Error code 1 (invalid function number) is returned if the IBM PC Local Area Network program is not loaded.

Error codes are returned in AX. Issue function call 59H (Get Extended Error) for additional information about the error class, suggested

action, and location. Refer to "Responding to Errors" on page B-6 and "Extended Error Codes" on page B-7 for more information on the codes returned from function call 59H.

62H —
Get Program Segment Prefix Address

Purpose
Returns the program prefix address.

Example

```
        MOV     AH,62H          ; Function Call -
                                ; Get Program Segment Prefix Address
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     PSPSeg,BX       ; Save PSP address

----

PSPSeg          DW      ?       ; Segment address of my PSP
```

Comments
The internal PSP address for the currently executing process is returned in BX.

65H —
Get Extended Country Information

Purpose

Returns extended country information.

Example

```
; To get information

      MOV     AH,65H          ; Function Call -
                              ; Get extended country information
      MOV     AL,InfoID       ; Data/function requested
                              ; (1, 2, 4, 6 or 7).
      MOV     BX,CodePage     ; Set desired code page
                              ; (-1=current, Set by function call 6602H).
      MOV     CX,SizeBuffer   ; Maximum data to return
                              ; (must be >= 5)
      MOV     DX,CountryID    ; Set desired Country ID
                              ; (-1=current, set by function call 38H).
      MOV     DI,SEG Buffer   ; Information return buffer
      MOV     DS,DI
      MOV     DI,OFFSET Buffer
      INT     21H             ; Issue request to DOS
      JC      Error           ; Error code in AX

-----

Buffer        LABEL   BYTE
; Format depends on AL value
; AL=1 : Extended Country Information

SizeBuffer    DW      ?       ; Size of data block to return
InfoID        DB      ?       ; Type of info to get
CIinfoSize    DW      ?       ; amount of data that follows
                              ; (limited by CX input)
CountryID     DW      ?       ; Selected CountryID
CodePage      DW      ?       ; Selected Code Page
; See function call 38H for the format of the remainder of this buffer
```

```
---

; AL=2 : Upper Case Table
               DB      2               ; Indicates Upper Case Table
UpperCase@     DD      ?               ; Address of Upper Case Table

; AL=4 : File Upper Case Table
               DB      4               ; Indicates File Upper Case Table
UpperCase@     DD      ?               ; Address of File Upper Case Table

; AL=6 : Collating Table
               DB      6               ; Indicates Collate Table
Collate@       DD      ?               ; Address of Collate Table

; AL=7 : DBCS Vector Table
               DB      7               ; Indicates DBCS Vector Table
DBCS@          DD      ?               ; Address of DBCS Vector Table
```

Comments

On entry, DX contains the ID of the country for which the extended information is needed. AL contains the ID value for the country.

- If the country code and code page do not match, or if either one or both are invalid, an error code of 2 (file not found) is returned in AX.
- The size requested in CX must be 5 or greater. If it is less than 5, an error code of 1 is returned in AX.
- If the amount of information returned is greater than the size requested in CX, it is ended and no error is returned in AX.

Note: For further information on the country information, see function call 38H (Get or Set Country Dependent Information).

The NLSFUNC DOS extension must be installed to get information for countries other than the Current Country.

The uppercase table and the filename uppercase tables are 130 bytes long, consisting of a length field (2 bytes), followed by 128 uppercase values for the upper 128 ASCII characters. They have the following layout:

```
Tsize          DW      128             ; Table Size
Table          DB      128 DUP(?)      ; Upper case versions of 80H to FFH
```

65H —
Get Extended Country Information

The following formula can be used to determine the address of an uppercase equivalent for a lowercase character (ASCII_in) in the uppercase table or the filename uppercase table.

Example

```
ASCII_in -(256-table_len)+table_start= address of ASCII_out
```

Where

```
ASCII_in    = character to be generated

table_len   = length of list of uppercase
values (2 bytes)

table_start = starting address of uppercase
table (4 bytes)

ASCII_out   = uppercase value for ASCII_in
```

If the value of ASCII_in is equal to or greater than (256-table_len), there is an uppercase equivalent for ASCII_in in the table. If it is lower than (256-table_len), no uppercase equivalent exists in the table.

The collate table is 258 bytes long, consisting of a length field (2 bytes) followed by 256 ASCII values, in the appropriate order. It has the following layout:

```
Tsize       DW      256             ; Table Size
Table       DB      256 DUP(?)      ; Sort Weights for 00H to FFH
```

The DBCS vector is variable in length, consisting of a length field (two bytes) followed by one or more pairs of bytes. It has the following layout:

```
Tsize   DW    Nx2           ;List size
1       DB    Start,end     ;DBCS vector 1
2       DB    Start,end     ;DBCS vector 2
:
N       DB    Start,end     ;DBCS vector n
        DB    0,0           ;End marker
```

Purpose

This function gets or sets the code page for the current country.

```
        MOV     AX,6601H        ; Function Call -
                                ; Get Global Code Page
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     GlobalCP,BX     ; Save Global Code Page
        MOV     SystemCP,DX     ; Save DOS System Code Page

      or

        MOV     AX,6602H        ; Function Call -
                                ; Set Global Code Page
        MOV     BX,GlobalCP     ; New Global Code Page
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

----

GlobalCP        DW      ?       ; Current Code Page of DOS Country Information
SystemCP        DW      ?       ; Code Page of DOS messages
                                ; Often the default Code Page for the
                                ; current country
```

Comments

DOS 4.00 moves the new code page data from the COUNTRY.SYS file to a resident country buffer area. DOS 4.00 uses the new code page to perform a Select to all attached devices that are set up for code page switching, (that is, have a code page switching device driver specified in CONFIG.SYS). If any device fails to be selected, an error code of 65 is returned in AX. The code page must be recognizable by the current country, and DOS 4.00 must be able to open and read from the country information file. Otherwise, the carry flag will be set on return and AX will contain 02 (file not found).

Note: NLSFUNC must be installed to use this function call, and all the devices must be prepared in order for the Select function to be successful.

67H —
Set Handle Count

Purpose
Permits more than 20 open files per process.

Example

```
EntryPoint:
        MOV     AH,62H          ; Function Call -
                                ; Get PSP Address
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     ES,BX           ; Set Segment
        MOV     AH,4AH          ; Function Call -
                                ; Set Memory Block Size
        MOV     BX,paragraphs   ; Set Size
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     AH,67H          ; Function Call - Set Handle Count
        MOV     BX,NewHandles   ; Set new handle count
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

  ----

NewHandles      DW      ?                           ; Number of Handles needed
                                                    ; by this DOS 4.00 process
ListSize        EQU     (Endofprogram - EntryPoint) ; Number of bytes
Paragraphs      EQU     (ListSize / 10H)            ; Number of paragraphs
End_of_program  LABEL   BYTE                        ; Last used position for
                                                    ;  this program
```

Comments
The maximum number of file handles allowed for this interrupt is 64KB. If the the specified number of allowable handles is less than the current number allowed, the specified number will become current only after all the handles above the specified number have been closed. If the specified number is less than 20, the number is assumed to be 20. Data base applications can use this function to reduce the need to swap handles.

You must release memory for DOS 4.00 to contain the extended handle list. You can do this by using the SET BLOCK (4AH) function call.

Purpose

Causes all buffered data for a file to be written to the device. This function can be used instead of a close-open sequence.

Example

```
        MOV     AH,68H          ; Function Call - Commit File
        MOV     BX,Handle       ; Select file
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

----

Handle          DW      ?       ; Handle from previous Open or Create
```

Comments

Commit File provides a faster and more secure method of committing data in multi-user environments such as the IBM PC Local Area Network.

6CH —
Extended Open/Create

Purpose
Optionally opens and creates a file.

Example

```
    MOV     AH,6CH              ; Extended open
    MOV     AL,0                ; Reserved
    MOV     BX,MODE             ; Open mode
                                ; Format : 0WF00000ISSS0AAA
                                ; AAA=Access code 0=Read
                                ;                 1=Write
                                ;                 2=Read/Write
                                ; SSS=Sharing mode 0=Compatibility
                                ;                  1=DenyRead/Write
                                ;                  2=DenyWrite
                                ;                  3=DenyRead
                                ;                  4=DenyNone

                                ; I 0=Pass handle to chile, 1=No inherit
                                ; F 0=INT 24H, 1=Return error
                                ;  on this open and any I/O to this handle
                                ; W 0=No commit, 1=Auto-Commit on write
    MOV     CX,ATTR             ; Create attribute (ignored if open)
    MOV     DX,FLAG             ; Function control, Format=0000000NNNNEEEE
                                ; NNNN=Does not exist action
                                ;   0=Fail, 1=Create
                                ; EEEE=Exists action
                                ;   0=Fail, 1=open, 2=Replace/Open

    MOV     SI,SEG file_name    ; Name to open or create
    MOV     DS,SI
    MOV     SI,OFFSET file_name
    INT     21H
    JC      ERROR
                                ; AX=Handle
                                ; CX=Action taken code
                                ; 1=File opened
                                ; 2=File created/opened
                                ; 3=File replaced/opened

;----
   Mode     DW        ?        ; Open mode bit
                               ; Definitions
   Attr     DW        ?        ; File attributes
   Flag     DW        ?        ; Function definition
   File_name    64 DUP (0)
```

Comments

Function 6CH combines the functions currently available with OPEN, CREATE and a CREATE NEW.

If F is 1, the critical error handler (Interrupt 24) is disabled for the handle returned by Extended Open. Any I/O issued to this handle will never generate the critical error but only the extended error.

If F is 0, no actions are taken.

If W is 1, any disk write using the handle returned by Extended Open will accompany with a commit call (see Interrupt 21H (AL=68H)).

If W is 0, no actions are taken.

Appendix C. I/O Control for Devices (IOCtl)

Purpose

Sets or gets device information associated with open device handles, or sends control strings to the device handle or receives control strings from the device handle.

Comments

The following function values are allowed in AL:

AL = 00H	Get device information (returned in DX).
AL = 01H	Set device information (determined by DX). DH must be 0 for this call.
AL = 02H	Read from character device
AL = 03H	Write to character device
AL = 04H	Read from block device
AL = 05H	Write to block device
AL = 06H	Get input status
AL = 07H	Get output status
AL = 08H	Determine if a particular block device is removable
AL = 09H	Determine if a logical device is local or remote
AL = 0AH	Determine if a handle is local or remote
AL = 0BH	Change sharing retry count
AL = 0CH	Issue handle generic IOCtl request
AL = 0DH	Issue block device generic IOCtl request
AL = 0EH	Get logical drive
AL = 0FH	Set logical drive

IOCtl can be used to get information about devices. You can make calls on regular files, but only function values 00H, 06H, and 07H are defined in that case. All other calls return an "Invalid Function" error.

Function values 00H to 08H are not supported on network devices. Function value 0BH requires the file sharing command to be loaded (SHARE).

Many of the function calls return the carry flag clear if the operation was successful. If an error condition was encountered, the carry flag is set; AX contains an extended error code. (See "Extended Error Codes" on page B-7 in *Appendix B* for an explanation). An explana-

tion of the error codes for call 440CH can be located beginning on page C-14 in this chapter. Information about the error, such as the *error class*, *location*, and recommended *action*, is obtained by issuing the 59H (Get Extended Error) function call.

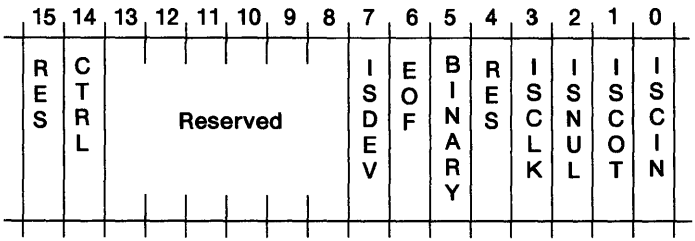
Calls AL = 00H and AL = 01H

Purpose
Sets or gets device information.

Example

```
; To Get Device Information

        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,0            ; Indicate get device information
        MOV     BX,Handle       ; Select device
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     DevInfo,DX      ; Save device information

; To Set Device Information

        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,1            ; Indicate set device information
        MOV     BX,Handle       ; Select device
        MOV     DX,DevInfo      ; Device information to set
        XOR     DH,DH           ; All DH bits must be off
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

-----

DevInfo         DW      ?       ; Device information
Handle          DW      ?       ; Handle to open device
```

Comments
The bits of DevInfo are defined as follows:



ISDEV = 1 if this channel is a device.

= 0 if this channel is a disk file (bits 8 through 15 are 0 in this case).

Bits 8 through 15 of DX correspond to the upper 8 bits of the device driver attribute word.

If ISDEV = 1

EOF = 0 if end-of-file on input.

BINARY = 1 if operating in binary mode (no checks for Ctrl-Z).

= 0 if operating in ASCII mode (checking for Ctrl-Z as end-of-file).

ISCLK = 1 if this device is the clock device.

ISNUL = 1 if this device is the null device.

ISCOT = 1 if this device is the console output.

ISCIN = 1 if this device is the console input.

CTRL = 0 if this device cannot process control strings via calls AL=02H, AL=03H, AL=04H, and AL=05H

CTRL = 1 if this device can process control strings via calls AL=02H and AL=03H. Note that this bit cannot be set by function call 44H.

If ISDEV = 0

EOF = 0 if channel has been written. Bits 0-5 are the block device number for the channel (0 = A, 1 = B, ...). Bits 15, 8-13, 4 are reserved and should not be altered.

Note: DH must be 0 for call AL=01H.

Calls AL = 02H, AL = 03H

Purpose

These two calls allow control strings to be sent or received from a character device.

Example

```
; To Read a Control String from a Character Device

      MOV     AH,44H            ; Function Call - IOCtl
      MOV     AL,2              ; Indicate IOCtl read
      MOV     BX,Handle         ; Select device
      MOV     CX,SIZE Buffer    ; Set size to read
      MOV     DI,SEG Buffer     ; Address I/O buffer
      MOV     DS,DI
      MOV     DX,OFFSET Buffer ; DS:DX points to I/O buffer
      INT     21H               ; Issue request to DOS
      JC      Error             ; Error code in AX
      MOV     Count,AX          ; Save data read count

; To Write a Control String to a Character Device

      MOV     AH,44H            ; Function Call - IOCtl
      MOV     AL,3              ; Indicate IOCtl write
      MOV     BX,Handle         ; Select device
      MOV     CX,SIZE Buffer    ; Set size to write
      MOV     DI,SEG Buffer     ; Address I/O buffer
      MOV     DS,DI
      MOV     DX,OFFSET Buffer ; DS:DX points to I/O buffer
      INT     21H               ; Issue request to DOS
      JC      Error             ; Error code in AX
      MOV     Count,A X         ; Save data written count

-----

Handle  DW    ?                 ; Handle to open device
Buffer  DB    N DUP(?)          ; I/O buffer
Count   DW    ?                 ; Actual I/O data transfer count
```

Comments

These are the Read and Write calls for a character device. An "Invalid Function" error is returned if the CTRL bit is 0.

44H —
I/O Control for Devices (IOCtl)

Calls AL = 04H, AL = 05H

Purpose
These two calls allow arbitrary control strings to be sent or received from a block device.

Example

```
; To Read a Control String from a Block Device

      MOV     AH,44H           ; Function Call - IOCtl
      MOV     AL,4             ; Indicate IOCtl read
      MOV     BL,Drive         ; Select drive
      MOV     CX,SIZE Buffer   ; Set Size to read
      MOV     DI,SEG Buffer    ; Address I/O buffer
      MOV     DS,DI
      MOV     DX,OFFSET Buffer ; DS:DX points to I/O buffer
      INT     21H              ; Issue request to DOS
      JC      Error            ; Error code in AX
      MOV     Count,AX         ; Save data read count

; To Write a Control String to a Character Device

      MOV     AH,44H           ; Function Call - IOCtl
      MOV     AL,5             ; Indicate IOCtl write
      MOV     BL,Drive         ; Select drive
      MOV     CX,SIZE Buffer   ; Set Size to write
      MOV     DI,SEG Buffer    ; Address I/O buffer
      MOV     DS,DI
      MOV     DX,OFFSET Buffer ; DS:DI points to I/O buffer
      INT     21H              ; Issue request to DOS
      JC      Error            ; Error code in AX
      MOV     Count,AX         ; Save data written count

-----

Drive         DB   ?           ; Drive (0=current, 1=A:, 2=B:, ...)
Buffer        DB   N DUP(?)    ; I/O buffer
Count         DW   ?           ; Actual I/O data transfer count
```

Comments
These are the Read and Write calls for a block device. The drive number is in BL for these calls. An "Invalid Function" error is returned if the CTRL bit is 0. An "Access-Denied" code is returned if the drive is invalid.

Calls AL = 06H and AL = 07H

Purpose

These calls allow you to check if a handle is ready for input or output.

Example

```
; To Get Input Device Status

        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,6            ; Indicate IOCtl input status
        MOV     BX,Handle       ; Select device
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     Status,AL       ; Save status

; To Get Output Device Status

        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,7            ; Indicate IOCtl output status
        MOV     BX,Handle       ; Select device
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     Status,AL       ; Save status

-----

Handle          DW      ?       ; Handle to open device
Status          DB      ?       ; Status
                                ; for a file:
                                ;   00H = At End of File
                                ;   FFH = Not at End of File
                                ; for a device:
                                ;   00H = Not ready
                                ;   FFH = Ready
```

Comments

If used for a file, AL always returns F2H until end-of-file is reached, then always returns 00H unless the current file position is changed through call 42H. When used for a device, AL returns FFH for ready or 0 for not ready.

44H —
I/O Control for Devices (IOCtl)

Call AL = 08H

Purpose
This call allows you to determine if a device can support removable media.

Example

```
        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,8            ; Indicate IOCtl is removable
        MOV     BL,Drive        ; Select drive
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        MOV     Dtype,AX        ; Save type

-----

Drive           DB      ?       ; Drive (0=current, 1=A:, 2=B:, ...)
Dtype           DW      ?       ; Drive type
                                ;   0 = Drive is removable
                                ;   1 = Drive is fixed
                                ; 0FH = Drive not valid
```

Comments
If the value returned in AX is 0, the device is removable. If the value is 1, the device is fixed. The drive number should be placed in BL. If the value in BL is invalid, an "Access-Denied" is returned. For network devices, the error "Invalid Function" is returned.

Call AL = 09H

Purpose

This call allows you to determine if a logical device is associated with a network directory.

Example

```
        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,9            ; Indicate IOCtl is remote drive
        MOV     BL,Drive        ; Select drive
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        TEST    DX,1000H        ; See if local/remote
        JNZ     Is_Remote       ; Drive is remote

-----

Drive           DB      ?       ; Drive (0=current, 1=A:, 2=B:, ...)
```

Comments

On entry, BL contains the drive number of the block device you want to check (0 = default, 1 = A, 2 = B, and so forth). The value returned in DX indicates whether the device is local or remote. Bit 12 is set for remote devices (1000H). Bit 12 is not set for local devices. The other bits in DX are reserved. If disk redirection is paused, the function returns with bit 12 not set.

44H —
I/O Control for Devices (IOCtl)

Call AL = 0AH

Purpose

This call allows you to determine if a handle is for a local device or a remote device across the network.

Example

```
        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,0AH          ; Indicate IOCtl is remote handle
        MOV     BX,Handle       ; Select device/file
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        TEST    DX,8000H        ; See if local/remote
        JNZ     Is_Remote       ; Drive is remote

-----

Handle          DW      ?       ; Handle to open file or device
```

Comments

For remote devices, bit 15 is set (8000H). The handle should be placed in BX. Bit 15 is not set for local devices.

Call AL = 0BH

Purpose
Controls retries on sharing and lock resource conflicts.

Example

```
        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,0BH          ; Indicate IOCtl set retry counts
        MOV     CX,NumLoops     ; Set number of loops
        MOV     DX,NumRetries   ; Set number of retries
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

-----

NumLoops        DW      ?       ; Number of times to execute loop below
NumRetries      DW      ?       ; Number of times to retry on error
```

Comments
All sharing and lock conflicts are automatically retried a number of times before they are returned as a DOS 4.00 error or critical error. You can select the number of retries and the delay time between retries. On input, CX contains the number of times to execute a delay loop, and DX contains the number of retries. The delay loop consists of the following sequence:

```
XOR     CX,CX
LOOP    $           ;spin 64K times
```

If this call is not issued, DOS 4.00 uses delay = 1 and retries = 3 as the defaults for CX and DX. If you expect your application to cause sharing or lock conflicts on locks that are in effect for a short period of time, you may want to increase the values for CX and DX to minimize the number of errors actually returned to your application.

44H —
I/O Control for Devices (IOCtl)

Call AL = 0CH

Purpose

This generic IOCtl function uses an open device handle to request a device driver to perform code page switching or to get/set device information.

Example

```
        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,0CH          ; Indicate file handle generic IOCtl request
        MOV     BX,Handle       ; Select device/file
        MOV     CH,Category     ; Set device type
        MOV     CL,Function     ; Set function
        MOV     DI,SEG Packet   ; Address subfunction parameter packet
        MOV     DS,DI
        MOV     DX,OFFSET Packet ; DS:DX points to parameter packet
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX

-----

Handle          DW      ?       ; Handle to open file or device
Category        DB      ?       ; Type of device
                                ; 0 - Unknown (if device type not known)
                                ; 1 - a COMx device
                                ; 3 - CON
                                ; 5 - a LPTx device
Function        DB      ?       ; Function within category
                                ; For category 3 & 5:
                                ;   4CH = Prepare start
                                ;   4DH = Prepare end
                                ;   4AH = Select (Set)  code page
                                ;   6AH = Query  (Get) selected code page
                                ;   6BH = Query prepare list
                                ; For Category 3:
                                ;   5FH = Set display information
                                ;   7FH = Get display information
```

Prepare Start: When CL=4CH, the parameter block, pointed to by DS:DX, has the following layout:

```
Packet          Label              Word
  PS_PACKET     STRUC            ; Prepare start packet
  PS_FLAGS      DW      0        ; Control flags
                                      ; Bit 0 = 0 : Download prepare
                                      ; BIT 0 = 1 : Cartridge prepare
                                      ; Others reserved (set to 0)
  PS_LENGTH     DW      (n+1)*2  ; Length of rest of packet in bytes
  PS_NUMCP      DW      n        ; Number of code pages
  PS_CP1        DW      ?        ; Code page 1
                    .
                    .
                    .
                    .
  PS_CPn        DW      ?        ; Code page n
  PS_PACKET     ENDS
```

Notes:

1. Setting any PS_CPn to -1 tells the device driver not to change the code page value for that position. Any other value is a code page to be prepared.
2. n is the number of additional code pages specified in the DEVICE= command in CONFIG.SYS. The value for n can be up to 12.
3. For cartridge-prepares set the PS_FLAGS field to 1.

A Prepare Start request begins the preparation of a code page. It is followed by writing data defining the code page font to the device driver using one or more IOCtl write control string calls (AX=4403H). It is assumed that this information will be downloaded to the device. The stream is ended by a Prepare End. The format of the stream is device dependent.

If the information is lost (due to a system failure or power-off), you do not have to rewrite the prepared code page. Requesting a "refresh" operation by issuing a Prepare Start to the device driver with all code page values (PS_CPn) set to a negative one (-1), restores the most recently prepared code page information to the device. You must follow this operation immediately with a Prepare End.

If no data is written for a prepare operation, the driver interprets the newly prepared code page(s) as a hardware code page. This allows devices that support user changeable hardware fonts (usually in cartridges) to be supported.

No prepare is needed for hardware-defined code pages.

Prepare Start Error Codes

Code	Meaning
01	Invalid function number
22	Unknown command
27	Code page conflict (used for KEYBxx mismatch)
29	Device error
31	Device driver does not have copy of code page to download to device

Write Error Codes

Code	Meaning
27	Device not found in file, or code page not found in file
29	Device error
31	File contents not a font file, or file contents structure damaged

Prepare End: When CL=4DH the parameter block, pointed to by DS:DX, has the following layout:

```
Packet          Label       Word
  PE_PACKET     STRUC           ; Prepare end packet
  PE_LENGTH     DW      2       ; Length of packet in bytes
  PE_RESV1      DW      0       ; (Reserved, must be 0)
  PE_PACKET     ENDS
```

Prepare End Error Codes

Code	Meaning
19	Bad data read from font file
31	No prepare start

Select/Query Selected Code Page: When CL = 4AH or 6AH the parameter block, pointed to by DS:DX, has the following layout:

```
PACKET LABEL WORD
 CP_PACKET     STRUC                ; Select/Query Selected packet
 CP_LENGTH     DW      2+(n+1)*2    ; Length of packet in bytes
 CP_CPID       DW      ?            ; Code page ID
 CP_VECTOR1    DB      ?,?          ; DBCS vector 1
                              .
                              .
                              .
 CP_VECTORn    DB      ?,?          ; DBCS Vector n
                       DB      0,0          ; End marker
CP_PACKET      ENDS
```

Select/Query Selected Code Page also includes the DBCS environment vector. Some device drivers may support only the code page value. As a result, you must check the returned length when using this call to determine if the DBCS information is present. Only the drivers supplied with the Asian version of DOS 4.00 provide this support.

Select Code Page Error Codes

Code	Meaning
26	Code page not prepared
27	Current keyboard does not support this code page
29	Device error

Query Selected Code Page Error Codes

Code	Meaning
26	No code page has been selected
27	Device error

Query Prepared List: When CL=6BH, the parameter block, pointed to by DS:DX, has the following layout:

```
PACKET              LABEL               WORD
  QL_PACKET     STRUC                ; Query list packet
  QL_LENGTH     DW((m+1)+(n+1))*2    ; Length of packet in bytes
  QL_NUMHWCP    DW      n            ; Number of hardware code pages
  QL_HWCP1      DW      ?            ; Hardware code page 1
                    .
                    .
                    .
                    .
  QL_HWCPn      DW      ?            ; Hardware code page n
  QL_NUMCP      DW      n            ; Number of prepared code pages
  QL_CP1        DW      ?            ; Prepared code page 1
                    .
                    .
                    .
                    .
  QL_CPm        DW      ?            ; Prepared code page n
  QL_PACKET     ENDS
```

Note: The device driver may return up to 12 code page values for each type of code page (hardware or prepared) so n can be up to 12, and m can be up to 12.

Query Prepared List Error Codes

Code	Meaning
26	No code pages have been selected
29	Device error

Get/Set Display Information: When CL = 5FH or 7FH, the parameter block, pointed to by DS:DX, has the following layout:

```
VP_PACKET     STRUC          ; Video parameters packet
VP_LEVEL      DB      0      ; Requested info level (set to 0 before IOCtl
                                    ; Call)
VP_RESV1      DB      0      ; (Reserved, must be 0)
VP_LENGTH     DW      14     ; Length of rest of packet in bytes
VP_FLAGS      DW      0      ; Control flags
                             ; Bit 0 = 0 : Intense colors
                             ; Bit 0 = 1 : Blink
                             ; Others reserved (set to 0)
VP_MODE       DB      ?      ; Video mode
                             ; 1 = Text
                             ; 2 = APA
                             ; Others reserved
VP_RESV2      DB      0      ; (Reserved. must be 0)
VP_COLORS     DW      ?      ; Number of colors (Mono=0)
VP_WIDTH      DW      ?      ; Display width in pixels (APA mode only)
VP_LENGTH     DW      ?      ; Display length in pixels (APA mode only)
VP_COLS       DW      ?      ; Display width in characters
VP_ROWS       DW      ?      ; Display length in characters

VP_PACKET     ENDS
```

44H —
I/O Control for Devices (IOCtl)

Call AL = 0DH

Purpose

This generic IOCtl function requests a block device driver to perform one of the following subfunctions:

- Get device parameters
- Set device parameters
- Read track on a logical device
- Write track on a logical device
- Format and verify track on a logical device
- Verify track on a logical device
- Get access flag status
- Set access flag status.

Example

```
        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,0DH          ; Indicate Block Device Generic IOCtl
        MOV     BL,Drive        ; Select Device/File
        MOV     CH,Category     ; Set Device Type
        MOV     CL,Function     ; Set Function
        MOV     DS,SEG Packet   ; Address Subfunction Parameter Packet
        MOV     DX,OFFSET Packet
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
-----

Drive           DB      ?       ; Drive (0=current, 1=A:, 2=B:, ...)
Category        DB      ?       ; Type of device
                                ; 8 - Block Device
Function        DB      ?       ; Function within Category
                                ; For Category 8:
                                ;   40H = Set device parameters
                                ;   60H = Get device parameters
                                ;   41H = Write track on logical device
                                ;   61H = Read track on logical device
                                ;   42H = Format and verify track on a
                                ;         logical device
                                ;   62H = Verify track on a logical device
                                ;   47H = Set access flag
                                ;   67H = Get access flag
```

Note: Functions 43H through 46H and functions 63H through 66H are reserved for the system.

Comments

CH contains the major code (08H for all functions) and CL contains the minor code (function).

Get or Set Device Parameters

To Get Device Parameters, set CL = 60H.

To Set Device Parameters, set CL = 40H.

When CL = 60H or CL = 40H, the parameter block has the following field layout:

```
Packet                  Label       Byte
 A_deviceParameters     STRUC
 SpecialFunctions       DB      ?
 DeviceType             DB      ?
 DeviceAttributes       DW      ?
 NumberOfCylinders      DW      ?
 MediaType              DB      ?
 DeviceBPB              a_BPB   <>
 TrackLayout            a_TrackLayout <>
 A_deviceParameters     ends
```

An explanation of each field in the parameter block is given in the pages that follow.

SpecialFunctions Field

This 1-byte field is used to further define the Get and Set Device Parameters functions.

For the Get Device Parameters function, bit 0 of the **SpecialFunctions** field has the following meaning:

Bit 0 = 1 Return the BPB that BUILD BPB would return.
= 0 Return the default BPB for the device.

Note: All other bits must be off.

For the Set Device Parameters function bits 0, 1, and 2 of the **SpecialFunctions** field are used.

These bits have the following meanings when CL = 40H.

Bit 0 =1 All subsequent BUILD BPB requests return **DeviceBPB**. If another Set Device request is received with bit 0 reset, BUILD BPB returns the actual media BPB.
=0 Indicates that the **DeviceBPB** field contains the new default BPB for this device. If a previous Set Device request set this bit on, the actual media BPB is returned. Otherwise, the default BPB for the device is returned by BUILD BPB.

Bit 1 =1 Ignore all fields in the Parameter Block except the **TrackLayout** field.
=0 Read all fields of the parameter block.

Bit 2 =1 Indicates that all sectors in the track are the same size and all sector numbers are between 1 and *n* (where *n* is the number of sectors in the track.)
=0 Indicates that all sectors in the track may not be the same size.

Notes:

1. All other bits must be reset.
2. Set bit 2 for normal track layouts. Format Track can be more efficient if bit 2 is set.
3. Setting bits 0 and 1 at the same time is invalid and should be considered an error.

DeviceType Field

This 1-byte field describes the physical device type. Device type is not set by IOCtl but is received from the device.

The values in this field have the following meanings:

0 = 320/360 KB 5.25 inch
1 = 5.25 inch, 1.2 MB
2 = 3.5 inch, 720 KB
3 = 8-inch single-density
4 = 8-inch double-density
5 = Fixed disk
6 = Tape drive
7 = Other.

DeviceAttributes Field

A 1-word field that describes the physical attributes of the device. Device attributes are not set by IOCtl but are received from the device driver.

Only bits 0 and 1 of this field are used. They have the following meanings:

Bit 0 =1 media is not removable.
=0 media is removable.
Bit 1 =1 diskette changeline is supported.
=0 diskette changeline is not supported.

Bits 2 – 15 are reserved.

NumberOfCylinders Field

This field indicates the maximum number of cylinders supported on the physical device, independent of the media type. The information in this field is not set by IOCtl, but is received from the device driver.

MediaType Field

For multimedia drives, this field indicates which media is expected to be in the drive. This field is only meaningful for Set Device Parameters (CL = 40H) subfunction.

The **MediaType** field is used only when the actual media in the drive cannot otherwise be determined. Media type is dependent on device type.

Regardless of the device type, a value of 0 represents the default. For example, a 5.25-inch 1.2MB diskette drive is a multimedia drive. The media type is defined as follows:

0 = Quad density 1.2 MB (96 tpi) diskette

1 = Double density 320/360KB (48 tpi) diskette

The default media type for a 1.2MB drive is a quad density 1.2 MB diskette.

DeviceBPB Field

For the Get Device Parameters function:

- If bit 0 of the **SpecialFunctions** field is set, the device driver returns the BPB that BUILD BPB would return.
- If bit 0 of the **SpecialFunctions** field is not set, the device driver returns the default BPB for the device.

For the Set Device Parameters function:

- If bit 0 of the **SpecialFunctions** field is set, the device driver is requested to return the BPB from this field for all subsequent BUILD BPB requests until a Set Device Parameters request is received with bit 0 in the **SpecialFunctions** field reset.
- If bit 0 is not set, the BPB contained in this field becomes the new default BPB for the device.

The **DeviceBPB** field has the following format:

```
a_BPB              STRUC
BytesPerSector     DW      ?
SectorsPerCluster  DB      ?
ReservedSectors    DW      ?
NumberOfFATs       DB      ?
RootEntries        DW      ?
TotalSectors       DW      ?
MediaDescriptor    DB      ?
SectorsPerFAT      DW      ?
;
SectorsPerTrack    DW      ?
Heads              DW      ?
HiddenSectors      DD      ?
BigTotalSectors    DD      ?
Reserved           DB      6 Dup (0)
a_BPB              ENDS
```

TrackLayout Field

This is a variable length table indicating the expected layout of sectors on the media track.

DOS 4.00 device drivers do not keep a track layout table for each logical device. The global track table must be updated (by the Set

Device Parameters subfunction) when the attributes of the media change.

Note: The Set Device Parameters subfunction (CL=40H) modifies the track table regardless of how bit 1 of the **SpecialFunctions** field is set.

For Get Device Parameters, this field is not used. The track layout is used by subsequent Read/Write Track, Format/Verify Track and Verify Track functions.

The following example shows how this field is formatted:

```
Total sectors-------|SectorCount      DW  n

Sector  1-----------|SectorNumber_1   DW  1H
                    |SectorSize_1     DW  200H

Sector  2-----------|SectorNumber_2   DW  2H
                    |SectorSize_2     DW  200H

Sector  3-----------|SectorNumber_3   DW  3H
                    |SectorSize_3     DW  200H

Sector  4-----------|SectorNumber_4   DW  4H
                    |SectorSize_4     DW  200H
                          |
                          |
                          |
Sector  n-----------|SectorNumber_n   DW  n
                    |SectorSize_n     DW  200H
```

Note: All values are in hexadecimal.

The total number of sectors is indicated by the **SectorCount** field. Each sector number must be unique and in a range between 1 and n (sector count). As shown in the example above, the first sector number is 1 and the last sector number is equal to the sector count (n). If bit 2 of the **SpecialFunctions** field is set, all sector sizes, which are measured in bytes, must be the same. See the description of bit 2 under the **SpecialFunction** field.

Note: The **DeviceType**, **DeviceAttributes**, and **NumberOfSectors** fields should be changed only if the physical device has been changed.

44H —
I/O Control for Devices (IOCtl)

Read/Write Track on Logical Device

To read a track on a logical device, set CL = 61H.

To write a track on a logical device, set CL = 41H.

The parameter block has the following layout when reading or writing a track on a logical device.

```
Packet                  LABEL   BYTE
 a_ReadWriteTrackPacket STRUC
 SpecialFunctions       DB      ?
 Head                   DW      ?
 Cylinder               DW      ?
 FirstSector            DW      ?
 NumberOfSectors        DW      ?
 TransferAddress        DD      ?
 A_ReadWriteTrackPacket ENDS
```

Notes:

1. All bits in the **SpecialFunctions** field must be reset.
2. The value in the **FirstSector** field and the **NumberOfCylinders** field is 0-based. For example, to indicate sector 9, set the value to 8.

Format/Verify Track on Logical Drive (IOCtl Write)

To format and verify a track, set CL = 42H.

To verify a track, set CL = 62H.

The parameter block has the following layout when formatting a track or verifying a track on a logical drive.

```
PACKET                  LABEL   BYTE
 A_FormatPacket         STRUC
 SpecialFunctions       DB      ?
 Head                   DW      ?
 Cylinder               DW      ?
 A_FormatPacket         ENDS
```

On entry, bit 0 of the SpecialFunctions field has the following meanings:

Bit 0 = 1 Format status check call to determine if a combination of number-of-tracks and sectors-per-track is supported.

= 0 Format /Verify track call.

To determine if a combination of number-of-tracks and sectors- per-track is supported, a Set Device Parameters call must be issued with the correct BPB for that combination before issuing the Format Status call. The device driver can then return the correct code to indicate what is supported. The value returned in the SpecialFunctions field for a Format Status Check call are:

0 = This function is supported by the ROM BIOS. The specified combination of number-of-tracks and sectors-per-track is allowed for the diskette drive.

1 = This function is not supported by the ROM BIOS.

2 = This function is supported by the ROM BIOS. The specified combination of number-of-tracks and sectors-per-track is not allowed for the diskette drive.

3 = This function is supported by the ROM BIOS, but ROM BIOS cannot determine if the numbers-of-tracks and sectors-per-track are allowed because the diskette drive is empty.

To format a track:

1. Issue the Set Device Parameters function call.
2. Issue the Format Status Check function call to validate the number-of-tracks and sectors-per-track combination. Ignore the result if the value returned is 1, because the ROM BIOS does not support this function.
3. Issue the Format/Verify Track function call with the SpecialFunctions bit 0 reset for each track on the medium.

Get/Set AccessFlag Status

To get the access flag status of a fixed disk, set CL=67H.

To set the access flag status of a fixed disk, set CL=47H.

The parameter block has the following layout when getting or setting the access flag status of a fixed disk:

```
PACKET                  LABEL       BYTE
a_DiskAccess_Control    STRUC
SpecialFunctions        DB     0
DiskAccess_Flag         DB     ?    ; 0 = Disallow disk access
                                    ; Other value = allow disk access
a_DiskAccess_Control    ENDS
```

If the media has not been formatted or has an invalid boot record, the system will not allow disk I/O for the media. This ensures data integrity of fixed media. Since formatting a media is a special activity, and is needed to perform disk I/O for unformatted media, additional functions to control the disk access flag are necessary. A format utility should issue "Set the access flag status (CL=47H)" with DiskAccess_Flag = non-zero value to access the unformatted media. When every format operation is a success, leave the access flag status as it is to allow further disk I/O from general users. If format fails, issue "Set the access flag status (CL = 47H)" with DiskAccess_Flag = zero in order to block further media access. To get the current status of the system disk access flag, issue "Get the access flag status (CL = 67H)". If DiskAccess_Flag = zero, disk I/O is not allowed for the media.

Call AL = 0EH

Purpose

Allows the device driver to determine if more than one logical drive is assigned to a block device. When this call is issued, a drive number is passed in BL on input.

Example

```
        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,0EH          ; Indicate Logical Drive Check
        MOV     BL,Drive        ; Select Drive
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        CMP     AL,0            ; Only one drive letter for this device?
        JE      Single_Drive    ; Yes!
        MOV     ActiveDrive,AL  ; Save Active Drive info

-----

Drive           DB      ?       ; Drive (0=current, 1=A:, 2=B:, ...)
ActiveDrive     DB      ?       ; Current drive letter for this device
                                ; (1=A:, 2=B:, ...)
```

Comments

If the block device has more than one logical drive letter assigned to it, on output a drive number corresponding to the last drive letter that was used to reference the device is returned in AL. If only one drive letter is assigned to the device, 0 is returned in AL by this call.

44H —
I/O Control for Devices (IOCtl)

Call AL = 0FH

Purpose
This call requests the device driver to change the next logical drive letter that will be used to reference a block device.

Example

```
        MOV     AH,44H          ; Function Call - IOCtl
        MOV     AL,0FH          ; Indicate Set Logical Drive
        MOV     BL,Drive        ; Set Drive
        INT     21H             ; Issue request to DOS
        JC      Error           ; Error code in AX
        CMP     AL,0            ; Only one drive letter for this device?
        JE      Single_Drive    ; Yes!
        MOV     ActiveDrive,AL  ; Save Active Drive info
                                ; (should be the same as BL in)

-----

Drive           DB      ?       ; Drive (0=current, 1=A:, 2=B:, ...)
ActiveDrive     DB      ?       ; Current drive letter for this device
                                ; (1=A:, 2=B:, ...)
```

Comments
When copying diskettes on a drive whose physical drive number has more than one logical drive letter assigned to it (for example, copying on a single drive system), DOS 4.00 issues diskette swap prompts to tell you which logical drive letter is currently referencing the physical drive number. As the drive changes from source to target, DOS 4.00 issues the message:

```
"Insert diskette for drive X: and strike any key
when ready."
```

It is possible to avoid this message by issuing call AL = 0FH (Set Logical Drive).

To avoid the DOS 4.00 diskette swap message, set BL to the drive number that corresponds to the drive letter that will be referenced in the next I/O request.

Note: You can determine the last logical drive letter assigned to the physical drive number by issuing call AL = 0EH.

Because any block device can have logical drives, this call should be issued before all I/O operations involving more than one drive letter; otherwise, the DOS 4.00 message may be issued.

Appendix D. Expanded Memory Support

Expanded memory is memory addressable through a combination of an Expanded Memory Specification (EMS) device driver and an EMS-capable hardware adapter.

The table below describes the *Lotus/Intel/Microsoft (LIM) Expanded Memory Manager Specification Version 4.0* functions and shows you which ones are supported by DOS 4.00. For detailed information and guidelines on the use of these calls, refer to the LIM Specification.

LIM Spec.	INT 67H	Interface	DOS 4.00 Support	Description
1	AH = 40H	App	Yes	Get status
2	AH = 41H	App	Yes	Get page frame address
3	AH = 42H	App	Yes	Get unallocated page count
4	AH = 43H	App	Yes	Allocate pages
5	AH = 44H	App	Yes	Map/unmap handle page
6	AH = 45H	App	Yes	Deallocate pages
7	AH = 46H	App	Yes	Get EMM version
8	AH = 47H	App	Yes	Save page map
9	AH = 48H	App	Yes	Restore page map
10	Reserved			
11	Reserved			
12	AH = 4BH	App	Yes	Get EMM Handle count
13	AH = 4CH	App	Yes	Get EMM Handle pages
14	AH = 4DH	App	Yes	Get all EMM handle pages
15	AH = 4EH	App	Yes	Get/Set page map
16	AH = 4FH	App	Yes	Get/set partial page map

LIM Spec.	INT 67H	Interface	DOS 4.00 Support	Description
17	AH = 50H	App	Yes	Map/unmap multiple handle pages
18	AH = 51H	App	Yes	Reallocate pages
19	AH = 52H	App	No	Get/set handle attributes
20	AH = 53H	App	Yes	Get/set handle name
21	AH = 54H	App	Yes	Get handle directory
22	AH = 55H	App	Yes	Alter page map and jump
23	AH = 56H	App	Yes	Alter page map and call
24	AH = 57H	App	Yes	Move/exchange memory region
25	AH = 58H	App	Yes	Get mappable physical address array
26	AH = 59H	OS	Yes	Get expanded memory hardware information
27	AH = 5AH	App	Yes	Allocate new pages
28	AH = 5BH	OS	Yes	Alternate page map register set
29	AH = 5CH	App	Yes	Prepare expanded memory hardware for warm boot
30	AH = 5DH	App	Yes	Enable/disable OS/E function set

Index

Special Characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Numerics

Continued from inside front cover.

SUCH WARRANTIES ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

LIMITATION OF REMEDIES

IBM's entire liability and your exclusive remedy shall be as follows:

1) IBM will provide the warranty described in IBM's Statement of Limited Warranty. If IBM does not replace defective media or, if applicable, make the Program operate as warranted or replace the Program with a functionally equivalent Program, all as warranted, you may terminate your license and your money will be refunded upon the return of all of your copies of the Program.
2) For any claim arising out of IBM's limited warranty, or for any other claim whatsoever related to the subject matter of this Agreement, IBM's liability for actual damages, regardless of the form of action, shall be limited to the greater of $5,000 or the money paid to IBM, its Authorized Dealer or its approved supplier for the license for the Program that caused the damages or that is the subject matter of, or is directly related to, the cause of action. This limitation will not apply to claims for personal injury or damages to real or tangible personal property caused by IBM's negligence.
3) In no event will IBM be liable for any lost profits, lost savings, or any incidental damages or other consequential damages, even if IBM, its Authorized Dealer or its approved supplier has been advised of the possibility of such damages, or for any claim by you based on a third party claim.

Some states do not allow the limitation or exclusion of incidental or consequential damages so the above limitation or exclusion may not apply to you.

GENERAL

You may terminate your license at any time by destroying all your copies of the Program or as otherwise described in this Agreement.

IBM may terminate your license if you fail to comply with the terms and conditions of this Agreement. Upon such termination, you agree to destroy all your copies of the Program.

Any attempt to sublicense, rent, lease or assign, or, except as expressly provided herein, to transfer any copy of the Program is void.

You agree that you are responsible for payment of any taxes, including personal property taxes, resulting from this Agreement.

No action, regardless of form, arising out of this Agreement may be brought by either party more than two years after the cause of action has arisen except for breach of the provisions in the Section entitled "License" in which event four years shall apply.

This Agreement will be construed under the Uniform Commercial Code of the State of New York.

© IBM Corp. 1988
Printed in the
United States of America
All Rights Reserved

15F0256

DOS 4.00 Technical Reference Application Programming

Consult this Technical Reference for comprehensive information about the structure, facilities and program interfaces of IBM Disk Operating System (DOS) Version 4.00. It is designed for experienced DOS users, systems programmers and those who plan to develop their own applications.

3.5″ and 5.25″ diskettes

Programming Family

Warranty:
Media—three-month warranty
Program—three-month warranty

Program services:
Available after warranty—yes
Charges apply—no

Service expiration date:
8/31/89
No program services are available after this date.

®IBM is a registered trademark of International Business Machines Corporation.

International Business Machines Corporation

Armonk, New York 10504

IBM Program License Agreement

BEFORE OPENING THIS PACKAGE, YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS. OPENING THIS PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED AND YOUR MONEY WILL BE REFUNDED.

This is a license agreement and not an agreement for sale. IBM owns, or has licensed from the owner, copyrights in the Program. You obtain no rights other than the license granted you by this Agreement. Title to the enclosed copy of the Program, and any copy made from it, is retained by IBM. IBM licenses your use of the Program in the United States and Puerto Rico. You assume all responsibility for the selection of the Program to achieve your intended results and for the installation of, use of, and results obtained from, the Program.

The Section in the enclosed documentation entitled "License Information" contains additional information concerning the Program and any related Program Services.

LICENSE

You may:

1) use the Program on only one machine at any one time, unless permission to use it on more than one machine at any one time is granted in the License Information (Authorized Use);
2) make a copy of the Program for backup or modification purposes only in support of your Authorized Use. However, Programs marked "Copy Protected" limit copying;
3) modify the Program and/or merge it into another program only in support of your Authorized Use; and
4) transfer possession of copies of the Program to another party by transferring this copy of the IBM Program License Agreement, the License Information, and all other documentation along with at least one complete, unaltered copy of the Program. You must, at the same time, either transfer to such other party or destroy all your other copies of the Program, including modified copies or portions of the Program merged into other programs. Such transfer of possession terminates your license from IBM. Such other party shall be licensed, under the terms of this Agreement, upon acceptance of this Agreement by its initial use of the Program.

You shall reproduce and include the copyright notice(s) on all such copies of the Program, in whole or in part.

You shall not:

1) use, copy, modify, merge, or transfer copies of the Program except as provided in this Agreement;
2) reverse assemble or reverse compile the Program; and/or
3) sublicense, rent, lease, or assign the Program or any copy thereof.

LIMITED WARRANTY

Warranty details and limitations are described in the Statement of Limited Warranty which is available upon request from IBM, its Authorized Dealer or its approved supplier and is also contained in the License Information. IBM provides a three-month limited warranty on the media for all Programs. For selected Programs, as indicated on the outside of the package, a limited warranty on the Program is available. The applicable Warranty Period is measured from the date of delivery to the original user as evidenced by a receipt.

Certain Programs, as indicated on the outside of the package, are not warranted and are provided "AS IS."

SUCH WARRANTIES ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

LIMITATION OF REMEDIES

IBM's entire liability and your exclusive remedy shall be as follows:

1) IBM will provide the warranty described in IBM's Statement of Limited Warranty. If IBM does not replace defective media or, if applicable, make the Program operate as warranted or replace the Program with a functionally equivalent Program, all as warranted, you may terminate your license and your money will be refunded upon the return of all of your copies of the Program.
2) For any claim arising out of IBM's limited warranty, or for any other claim whatsoever related to the subject matter of this Agreement, IBM's liability for actual damages, regardless of the form of action, shall be limited to the greater of $5,000 or the money paid to IBM, its Authorized Dealer or its approved supplier for the license for the Program that caused the damages or that is the subject matter of, or is directly related to, the cause of action. This limitation will not apply to claims for personal injury or damages to real or tangible personal property caused by IBM's negligence.
3) In no event will IBM be liable for any lost profits, lost savings, or any incidental damages or other consequential damages, even if IBM, its Authorized Dealer or its approved supplier has been advised of the possibility of such damages, or for any claim by you based on a third party claim.

Some states do not allow the limitation or exclusion of incidental or consequential damages so the above limitation or exclusion may not apply to you.

GENERAL

You may terminate your license at any time by destroying all your copies of the Program or as otherwise described in this Agreement.

IBM may terminate your license if you fail to comply with the terms and conditions of this Agreement. Upon such termination, you agree to destroy all your copies of the Program.

Any attempt to sublicense, rent, lease or assign, or, except as expressly provided herein, to transfer any copy of the Program is void.

You agree that you are responsible for payment of any taxes, including personal property taxes, resulting from this Agreement.

No action, regardless of form, arising out of this Agreement may be brought by either party more than two years after the cause of action has arisen except for breach of the provisions in the Section entitled "License" in which event four years shall apply.

This Agreement will be construed under the Uniform Commercial Code of the State of New York.

Z125-3301-02 4/87

© IBM Corp. 1988
Printed in the United States of America
All Rights Reserved

6280254

9062802540001