



*Personal Computer
Programming Family*

DOS Technical Reference

First Edition (February 1985)

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.

Products are not stocked at the address below. Requests for copies of this publication and for technical information about IBM Personal Computer products should be made to your authorized IBM Personal Computer dealer, IBM Product Center, or your IBM Marketing Representative.

The following paragraph applies only to the United States and Puerto Rico: A Reader's Comment Form is provided at the back of this publication. If the form has been removed, address comments to: IBM Corporation, Personal Computer, P.O. Box 1328-C, Boca Raton, Florida 33432. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligations whatever.

© Copyright International Business Machines Corporation 1984, 1985

About This Book

Read This First

This book contains technical information pertaining to DOS versions 2.10, 3.00, and 3.10. Some information is specific to a version of DOS and does not apply to all versions.

This book covers topics for the experienced DOS user, system programmer, and application developer. It is assumed that you are familiar with the 8088 architecture.

Version Specific Information

Chapters that contain information that is specific to a version of DOS, contain a section called "Version Specific Information." This section identifies the information in the chapter that is for use with a particular version of DOS. Chapters that do not contain this section contain information that applies to DOS versions 2.10, 3.00, and 3.10.

How This Book is Organized

This book has 11 chapters.

Chapter 1 contains general technical information about DOS.

Chapter 2 contains detailed information about device drivers.

Chapter 3 contains detailed information about using extended screen and keyboard functions to control cursor positioning and to redefine keyboard keys.

Chapter 4 contains notes and considerations for proper file management.

Chapter 5 describes allocation of space on a disk.

Chapter 6 describes the system interrupts and function calls.

Chapter 7 describes control blocks and work areas, including a memory map, program segment prefix, and file control block.

Chapter 8 explains how to execute commands from within an application.

Chapter 9 contains technical information about DOS support of fixed disks.

Chapter 10 contains detailed information about .EXE file structure.

Chapter 11 contains information about DOS memory management.

Contents

Chapter 1. DOS Technical Information	1-1
Introduction	1-3
Version Specific Information	1-3
DOS Structure	1-4
The Boot Record	1-4
Read Only Memory (ROM) BIOS Interface	1-4
The DOS Program File	1-5
The Command Processor	1-5
DOS Initialization	1-7
Available DOS Functions	1-8
The Disk Transfer Area (DTA)	1-9
Error Trapping	1-10
Chapter 2. Installable Device Drivers	2-1
Introduction	2-3
Version Specific Information	2-3
Device Driver Format	2-4
Types of Devices	2-5
Character Devices	2-5
Block Devices	2-5
Device Header	2-6
Pointer to Next Device Header Field	2-6
Attribute Field	2-7
Pointer to Strategy and Interrupt Routines	2-9
Name/Unit Field	2-9
Creating a Device Driver	2-10
Installing Device Drivers	2-11
Installing Character Devices	2-12
Installing Block Devices	2-12
Request Header	2-14
Unit Code Field	2-14
Command Code Field	2-15
Status Field	2-16
Device Driver Functions	2-18
INIT	2-19

MEDIA CHECK	2-21
Media Descriptor Byte	2-23
BUILD BPB (BIOS Parameter Block)	2-26
INPUT or OUTPUT	2-29
NONDESTRUCTIVE INPUT NO WAIT	2-31
STATUS	2-32
FLUSH	2-33
OPEN or CLOSE (DOS 3.00 and 3.10)	2-34
REMOVABLE MEDIA (DOS 3.00 and 3.10)	2-35
The CLOCK\$ Device	2-36
Sample Device Driver	2-36

Chapter 3. Using Extended Screen and Keyboard

Control	3-1
Introduction	3-3
Control Sequences	3-3
Control Sequence Syntax	3-4
Cursor Control Sequences	3-6
Cursor Position	3-6
Cursor Up	3-7
Cursor Down	3-7
Cursor Forward	3-8
Cursor Backward	3-8
Horizontal and Vertical Position	3-9
Cursor Position Report	3-10
Device Status Report	3-10
Save Cursor Position	3-12
Restore Cursor Position	3-12
Erasing	3-13
Erase in Display	3-13
Erase in Line	3-13
Mode of Operation	3-14
Keyboard Key Reassignment	3-17

Chapter 4. File Management Notes	4-1
Introduction	4-3
Version Specific Information	4-3
File Management Functions	4-3
FCB Function Calls	4-5
Handle Function Calls	4-6
Special File Handles	4-8

ASCII and Binary Mode	4-9
File I/O in Binary Mode	4-10
File I/O in ASCII Mode	4-11
Number of Open Files Allowed	4-12
Restrictions on FCB Usage	4-12
Restrictions on Handle Usage	4-13
Allocating Space to a File	4-14
Chapter 5. DOS Disk Allocation	5-1
Introduction	5-3
Version Specific Information	5-3
The DOS Area	5-4
The Boot Record	5-4
DOS File Allocation Table (FAT)	5-5
How to Use the File Allocation Table for 12-Bit FAT Entries	5-8
How to Use the File Allocation Table for 16-Bit FAT Entries	5-9
DOS Disk Directory	5-10
Directory Entries	5-10
The Data Area	5-14
Chapter 6. DOS Interrupts and Function Calls	6-1
Introduction	6-7
Version Specific Information	6-7
DOS Registers	6-9
Extended ASCII Codes	6-11
Interrupts	6-12
20H Program Terminate	6-12
21H Function Request	6-13
22H Terminate Address	6-13
23H Ctrl-Break Exit Address	6-13
24H Critical Error Handler Vector ...	6-14
25H Absolute Disk Read	6-22
26H Absolute Disk Write	6-23
27H Terminate but Stay Resident	6-24
28H-2EH Reserved for DOS	6-25
2FH Multiplex Interrupt	6-26
30H-3FH Reserved for DOS	6-31
Function Calls	6-32
Listing of Function Calls	6-33
DOS Internal Stack	6-36
Error Return Information	6-36

ASCII Strings	6-44
Network Paths	6-45
Network Access Rights	6-45
File Handles	6-46
Using DOS Functions	6-47
00H Program Terminate	6-49
01H Keyboard Input	6-50
02H Display Output	6-51
03H Auxiliary Input	6-52
04H Auxiliary Output	6-53
05H Printer Output	6-54
06H Direct Console I/O	6-55
07H Direct Console Input Without Echo ..	6-56
08H Console Input Without Echo	6-57
09H Print String	6-58
0AH Buffered Keyboard Input	6-59
0BH Check Standard Input Status	6-60
0CH Clear Keyboard Buffer and Invoke a Keyboard Function	6-61
0DH Disk Reset	6-62
0EH Select Disk	6-63
0FH Open File	6-64
10H Close File	6-65
11H Search for First Entry	6-66
12H Search for Next Entry	6-68
13H Delete File	6-69
14H Sequential Read	6-70
15H Sequential Write	6-71
16H Create File	6-72
17H Rename File	6-73
19H Current Disk	6-74
1AH Set Disk Transfer Address	6-75
1BH Allocation Table Information	6-76
1CH Allocation Table Information for Specific Device	6-77
21H Random Read	6-78
22H Random Write	6-79
23H File Size	6-80
24H Set Relative Record Field	6-81
25H Set Interrupt Vector	6-82
26H Create New Program Segment	6-83
27H Random Block Read	6-84
28H Random Block Write	6-85

29H Parse Filename	6-86
2AH Get Date	6-88
2BH Set Date	6-89
2CH Get Time	6-90
2DH Set Time	6-91
2EH Set/Reset Verify Switch	6-92
2FH Get Disk Transfer Address (DTA) ...	6-93
30H Get DOS Version Number	6-94
31H Terminate Process and Remain Resident	6-95
33H Ctrl-Break Check	6-96
35H Get Vector	6-97
36H Get Disk Free Space	6-98
38H (DOS 2.10) Return Country Dependent Information	6-99
38H (DOS 3.00 and 3.10) Get or Set Country Dependent Information	6-101
39H Create Subdirectory (MKDIR)	6-106
3AH Remove Subdirectory (RMDIR) ...	6-107
3BH Change the Current Directory (CHDIR)	6-108
3CH Create a File (CREAT)	6-109
3DH (DOS 2.10) Open a File	6-110
3DH (DOS 3.00 and 3.10) Open a File ..	6-112
3EH Close a File Handle	6-122
3FH Read from a File or Device	6-123
40H Write to a File or Device	6-125
41H Delete a File from a Specified Directory (UNLINK)	6-127
42H Move File Read Write Pointer (LSEEK)	6-128
43H Change File Mode (CHMOD)	6-130
44H I/O Control for Devices (IOCTL) ..	6-132
45H Duplicate a File Handle (DUP)	6-139
46H Force a Duplicate of a Handle (FORCDUP)	6-140
47H Get Current Directory	6-141
48H Allocate Memory	6-142
49H Free Allocated Memory	6-143
4AH Modify Allocated Memory Blocks (SETBLOCK)	6-144
4BH Load or Execute a Program (EXEC)	6-145
4CH Terminate a Process (EXIT)	6-150
4DH Get Return Code of a Subprocess	

(WAIT)	6-151
4EH Find First Matching File (FIND FIRST)	6-152
4FH Find Next Matching File (FIND NEXT)	6-154
54H Get Verify Setting	6-155
56H Rename a File	6-156
57H Get/Set a File's Date and Time	6-158
59H (DOS 3.00 and 3.10) Get Extended Error	6-160
5AH (DOS 3.00 and 3.10) Create Unique File	6-162
5BH (DOS 3.00 and 3.10) Create New File	6-164
5CH (DOS 3.00 and 3.10) Lock/Unlock File Access	6-165
5E00H (DOS 3.10) Get Machine Name ..	6-168
5E02H (DOS 3.10) Set Printer Setup	6-169
5E03H (DOS 3.10) Get Printer Setup ...	6-170
5F02H (DOS 3.10) Get Redirection List Entry	6-171
5F03H (DOS 3.10) Redirect Device	6-173
5F04H (DOS 3.10) Cancel Redirection ..	6-176
62H (DOS 3.00 and 3.10) Get Program Segment Prefix Address (PSP)	6-178
Chapter 7. DOS Control Blocks and Work Areas .	7-1
Introduction	7-3
DOS Memory Map	7-4
DOS Program Segment	7-6
Program Segment Prefix	7-10
File Control Block	7-12
Standard File Control Block	7-13
Extended File Control Block	7-16
Chapter 8. Executing Commands From Within an Application	8-1
Introduction	8-3
Invoking a Command Processor	8-3
Chapter 9. Fixed Disk Information	9-1
Introduction	9-3
Fixed Disk Architecture	9-3
System Initialization	9-4

Boot/Record Partition Table	9-6
Fixed Disk Technical Information	9-8
Determining Fixed Disk Allocation	9-11
Chapter 10. EXE File Structure and Loading	10-1
Introduction	10-3
.EXE File Structure	10-3
The Relocation Table	10-5
Chapter 11. DOS Memory Management	11-1
Introduction	11-3
Control Block	11-3
Index	Index-1

Chapter 1. DOS Technical Information

Contents

Introduction	1-3
Version Specific Information	1-3
DOS Structure	1-4
The Boot Record	1-4
Read Only Memory (ROM) BIOS Interface ..	1-4
The DOS Program File	1-5
The Command Processor	1-5
DOS Initialization	1-7
Available DOS Functions	1-8
The Disk Transfer Area (DTA)	1-9
Error Trapping	1-10

Introduction

This chapter tells you about:

- DOS structure
- DOS initialization
- DOS functions
- Disk transfer area
- Error trapping

Version Specific Information

The following information in this chapter is specific to a version of DOS:

The Command Processor: For DOS 2.10, the transient portion of the command processor contains the EXEC routine that loads and executes external commands. For DOS versions 3.00 and 3.10, the resident portion of the command processor contains the EXEC routine.

DOS Structure

DOS consists of four components:

- The Boot Record
- The Read Only Memory BIOS Interface
- The DOS Program File (IBMDOS.COM)
- The command processor (COMMAND.COM)

The Boot Record

The boot record begins on track 0, sector 1, side 0 of every diskette formatted by the DOS FORMAT command. The boot record is placed on diskettes to produce an error message if you try to start up the system with a nonsystem diskette in drive A. For fixed disks, the boot record resides on the first sector of the DOS partition. All media supported by DOS use one sector for the boot record.

Read Only Memory (ROM) BIOS Interface

The file IBMBIO.COM is the interface module to the Read Only Memory (ROM) BIOS. IBMBIO.COM provides a low-level interface to the ROM BIOS device routines.

The DOS Program File

The DOS program is file `IBMDOS.COM`. It provides a high-level interface for user programs. `IBMDOS.COM` consists of file management routines, data blocking/deblocking for the disk routines, and a variety of built-in functions easily accessible by user programs.

When a user program calls these function routines, they accept high-level information by way of register and control block contents. For device operations, the functions translate the requirement into one or more calls to `IBMBIO.COM` to complete the request.

The Command Processor

The command processor, `COMMAND.COM`, consists of these parts:

1. A *resident* portion resides in memory immediately following `IBMDOS.COM` and its data area. This portion contains routines to process interrupts 22H (Terminate Address), 23H (Ctrl-Break Handler), and 24H (Critical Error Handling), as well as a routine to reload the transient portion if needed. For DOS 3.00 and 3.10, this portion also contains a routine to load and execute external commands, such as files with extensions of `.COM` or `.EXE`.

Note: When a program terminates, a checksum methodology determines if the program has caused the transient portion to be overlaid. If the transient portion is overlaid, it is reloaded.

All standard DOS error handling is done within this portion of `COMMAND.COM`. This includes displaying error messages and interpreting the replies of Abort, Retry, or Ignore. See the message "Disk error reading drive x" in Appendix A of the *DOS Reference*.

2. An *initialization* portion follows the resident portion and is given control during start-up. This portion contains the AUTOEXEC.BAT file processor setup routine. The initialization portion determines the segment address at which programs can be loaded. The initialization portion is overlaid by the first program COMMAND.COM loads because it's no longer needed.
3. A *transient* portion is loaded at the high end of memory. This is the command processor itself, containing all of the internal command processors and the batch file processor. For DOS 2.10, this portion also contains a routine to load and execute external commands, such as files with extensions of .COM or .EXE.

This portion of COMMAND.COM also produces the DOS prompt (such as A>), reads the command from the keyboard (or batch file), and executes the command. For external commands, it builds a command line and issues an EXEC function call to load and transfer control to the program.

Chapter 6 contains detailed information describing the conditions in effect when a program is given control by EXEC.

DOS Initialization

The system is initialized either by a system reset or by a power on. ROM BIOS first looks for the boot record on drive A. If the boot record is not found, ROM BIOS searches the active partition of the fixed disk. If it is not found there, ROM BIOS calls ROM BASIC. The following actions occur after a system initialization:

1. The boot record is read into memory and given control.
2. The boot record then checks the root directory to assure that the first two files are IBMBIO.COM and IBMDOS.COM. These two files must be the first two files, and they must be in that order (IBMBIO.COM first, with its sectors in contiguous order).
3. The boot record loads IBMBIO.COM into memory.
4. The initialization code in IBMBIO.COM loads IBMDOS.COM, determines equipment status, resets the disk system, initializes the attached devices, loads the installable device drivers, sets the low-numbered interrupt vectors, relocates IBMDOS.COM downward, and calls the first byte of DOS.
5. DOS initializes its internal working tables, initializes the interrupt vectors for interrupts 20H through 27H, and builds a Program Segment Prefix for COMMAND.COM at the lowest available segment. For DOS version 3.10, DOS initializes interrupt vectors for interrupts 0FH through 3FH.
6. IBMBIO.COM uses the EXEC function call to load and start the top-level command processor. The default command processor is COMMAND.COM.

Available DOS Functions

DOS provides a significant number of functions to user programs, all available through issuance of a set of interrupt and function calls. There are routines for keyboard input (with and without echo and Ctrl-Break detection), console and printer output, constructing file control blocks, memory management, date and time functions, and a variety of disk, directory, and file handling functions.

DOS provides two types of function calls that can be used for file management functions. They are:

- File control block (FCB) function calls
- Extended (Handle) function calls

See Chapter 4, “File Management Notes” for a description of FCB and Handle function calls. See Chapter 6, “DOS Interrupts and Function Calls” for detailed information on each individual call.

The Disk Transfer Area (DTA)

DOS uses an area in memory to contain the data for all file reads and writes that are performed with FCB function calls. This area in memory is called the *disk transfer area*. The disk transfer area (DTA) can also be called a *buffer*. This area can be at any location within the data area of your application program and should be set by your program.

Only one DTA can be in effect at a time, so your program must tell DOS what memory location to use *before* using any disk read or write functions. Use function call 1AH (Set Disk Transfer Address) to set the disk transfer address. Use function call 2FH (Get Disk Transfer Address) to get the disk transfer address. Refer to Chapter 6, "DOS Interrupts and Function Calls," for more information on these function calls. Once set, DOS continues to use that area for all disk operations until another function call 1AH is issued to define a new DTA. When a program is given control by COMMAND.COM, a default DTA large enough to hold 128 bytes is established at 80H into the program's Program Segment Prefix.

For file reads and writes that are performed with the extended function calls, there is no need to set a DTA address. Instead, specify a buffer address when you issue the read or write call.

Error Trapping

DOS provides a method by which a program can receive control whenever a disk or device read/write error occurs or when a bad memory image of the file allocation table is detected. When these errors occur, DOS executes an interrupt 24H (Critical Error Handler Vector), to pass control to the error handler. The default error handler resides in COMMAND.COM, but any program can establish its own by setting the interrupt 24H vector to point to the new error handler. DOS provides error information by using the registers and provides Abort, Retry, or Ignore support by using return codes. See "Error Return Information" in Chapter 6, "DOS Interrupts and Function Calls," for more information on error codes.

Chapter 2. Installable Device Drivers

Contents

Introduction	2-3
Version Specific Information	2-3
Device Driver Format	2-4
Types of Devices	2-5
Character Devices	2-5
Block Devices	2-5
Device Header	2-6
Pointer to Next Device Header Field	2-6
Attribute Field	2-7
Bit 15	2-7
Bit 14	2-7
Bit 13	2-8
Bit 11	2-8
Bit 3	2-8
Bit 2	2-9
Bits 0 and 1	2-9
Pointer to Strategy and Interrupt Routines ...	2-9
Name/Unit Field	2-9
Creating a Device Driver	2-10
Installing Device Drivers	2-11
Installing Character Devices	2-12
Installing Block Devices	2-12
Request Header	2-14
Unit Code Field	2-14
Command Code Field	2-15
Status Field	2-16

Device Driver Functions	2-18
INIT	2-19
MEDIA CHECK	2-21
Media Descriptor Byte	2-23
BUILD BPB (BIOS Parameter Block)	2-26
INPUT or OUTPUT	2-29
NONDESTRUCTIVE INPUT NO WAIT ..	2-31
STATUS	2-32
FLUSH	2-33
OPEN or CLOSE (DOS 3.00 and 3.10)	2-34
REMOVABLE MEDIA (DOS 3.00 and 3.10)	2-35
The CLOCK\$ Device	2-36
Sample Device Driver	2-36

Introduction

This chapter tells you how to:

- Format a device driver
- Create a device driver
- Install a device driver

This chapter also provides information on the types of device drivers, the request header, and the CLOCK\$ device.

The DOS device interface links the device drivers together in a chain. This allows you to add new device drivers for optional devices to DOS.

Version Specific Information

The following information in this chapter is specific to a version of DOS:

Attribute Field: Bit 11 (removable media) is for use with DOS versions 3.00 and 3.10.

Command Code Field: Command codes field values 13, 14, and 15 are for use with DOS versions 3.00 and 3.10.

Status Word Field: Error codes 0DH, 0EH, and 0FH are only returned when using DOS versions 3.00 and 3.10.

Device Driver Functions:

- DOS versions 3.00 and 3.10 support removable media.
- The Media Check device driver function returns “Error” as a possibility if you are using DOS versions 3.00 and 3.10. Also for DOS 3.00 and 3.10, Media Check returns a DWORD pointer to the volume ID if a disk change has occurred.
- Media descriptor byte F9H for 5 1/4 inch, 15 sector media is supported by DOS versions 3.00 and 3.10.
- For DOS 3.00 and 3.10, the Input or Output device driver function returns a DWORD pointer to the volume identification if an invalid disk change has occurred.
- The Open or Close device driver function is for use with DOS versions 3.00 and 3.10.
- The Removable Media device driver function is for use with DOS 3.00 and 3.10.

Device Driver Format

A device driver is a memory image file or an .EXE file that contains all of the code needed to implement the device. It has a special header at the front of it that identifies the file as a device driver, defines the strategy and interrupt entry points, and defines various attributes of the device.

Note: For device drivers, the memory image file must not use the ORG 100H. Because it does not use the program segment prefix, the device driver is simply loaded. Therefore, the memory image file must have an origin of 0 (ORG 0 or no ORG statement).

Types of Devices

There are two basic types of devices:

- Character devices
- Block devices

Character Devices

Character devices are designed to do character I/O in a serial manner like CON, AUX, and PRN. These devices have names like CON, AUX, CLOCK\$, and you can open channels (handles or FCBs) to do input and output to them. Because character devices have only one name, they can support only one device.

Block Devices

Block devices are the “fixed disk or diskette drives” on the system. They can do random I/O in pieces called blocks, which are usually the physical sector size of the disk. These devices are not *named* as the character devices are, and cannot be *opened* directly. Instead they are *mapped* by using the drive letters A, B, C, and so forth. Block devices can have units within them. In this way, a single block driver can be responsible for one or more disk or diskette drives. For example, the first block device driver can be responsible for drives A, B, C, and D. This means that it has four units defined and therefore takes up four drive letters. The position of the driver in the chain of all drivers determines the way the drive units and drive letters correspond. For example, if the device driver is the first block driver in the device chain, and it defines four units, then those units are A, B, C, and D. If the second block driver defines three units, then those units are E, F, and G. The limit is 26 devices with the letters A through Z assigned to the drives.

Device Header

A device driver requires a device header at the beginning of the file. Here is what the device header contains:

Field	Length
Pointer to next 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 Field

The device header field is a pointer to the device header of the next device driver. It is a double-word field that is set by DOS 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 is a word field that describes the attributes of the device driver to the system. The attributes are:

bit 15	= 1 character device 0 block device
bit 14	= 1 supports IOCTL 0 doesn't support IOCTL
bit 13	= 1 non-IBM format (block only) 0 IBM format
bit 11	= 1 supports removable media 0 doesn't support removable media
bits 10-4	= 0 these bits must be off because they are reserved by DOS
bit 3	= 1 current clock device 0 not current clock device
bit 2	= 1 current NUL device 0 not current NUL device
bit 1	= 1 current standard output device 0 not current standard output device
bit 0	= 1 current standard input device 0 not current standard input device

Bit 15

Bit 15 is the device type bit. Use bit 15 to tell the system if the device driver is a block or character device.

Bit 14

Bit 14 is the IOCTL bit. It is used for both character and block devices. Use bit 14 to tell DOS whether the device driver can handle control strings through the IOCTL function call (44H).

If a device driver cannot process control strings, it should set bit 14 to 0. This way DOS can return an error if an attempt is made through the IOCTL function

call to send or receive control strings to the device. If a device can process control strings, it should set bit 14 to 1. This way, DOS makes the calls to the IOCTL input and output device function to send and receive IOCTL strings.

The IOCTL functions allow data to be sent to and from the device without actually doing a normal read or write. In this way, the device driver can use the data for its own use (for example, setting a baud rate or stop bits, changing form lengths, and so forth). It is up to the device to interpret the information that is passed to it, but the information must not be treated as a normal I/O request.

Bit 13

Bit 13 is the non-IBM format bit. It is used for block devices only. It effects the operation of the the Get BPB (BIOS Parameter Block) device call.

Bit 11

Bit 11 is the open/close removable media bit. Use bit 11 to tell DOS if the device driver can handle removable media.

Bit 3

Bit 3 is the clock device bit. It is used for character devices only. Use bit 3 to tell DOS if your character device driver is the new CLOCK\$ device.

Bit 2

Bit 2 is the NUL attribute bit. It is used for character devices only. Use bit 2 to tell DOS if your character device driver is a NUL device. Although there is a NUL device attribute bit, you cannot reassign the NUL device. This is an attribute that exists for DOS so that DOS can tell if the NUL device is being used.

Bits 0 and 1

Bits 0 and 1 are the standard input/standard output bits. They are used for character devices only. Use these bits to tell DOS if your character device driver is the new standard input or standard output device.

Pointer to Strategy and Interrupt Routines

These two fields are the pointers to the entry points of the strategy and interrupt routines. They are word values, so they must be in the same segment as the device header.

Name/Unit Field

This is an 8-byte field that contains the name of a character device or the unit of a block device. For character devices, the name is left-justified and the space is filled to 8 bytes. For block devices, the number of units can be placed in the first byte. This is optional because DOS fills in this location with the value returned by the driver's INIT code.

Creating a Device Driver

To create a device driver that DOS can install, perform the following:

- Create a memory image file or an .EXE file with a device header at the start of the file.
- Originate the code (including the device header) at 0, not at 100H.
- Set the next device header field. Refer to “Pointer to Next Device Header Field” for more information.
- Set the attribute field of the device header. Refer to “Attribute Field” for more information.
- Set the entry points for the interrupt and strategy routines.
- Fill in the name/unit field with the name of the character device, or the unit number of the block device.

DOS always processes installable character device drivers before handling the default devices. So to install a new CON device, simply name the device CON. Be sure to set the standard input device and standard output device bits in the attribute field on a new CON device. The scan of the device list stops on the first match so the installable device driver takes precedence.

Note: Because DOS can install the driver anywhere in memory, care must be taken in any FAR memory references. You should not expect that your driver will always be loaded at the same place every time.

Installing Device Drivers

DOS installs new device drivers dynamically at boot time by reading and processing the DEVICE command in the CONFIG.SYS file. For example, if you have written a device driver called DRIVER1, to install it put this command in the CONFIG.SYS file:

```
device=driver1
```

DOS calls a device driver at its strategy entry point first, passing in a request header the information describing what DOS wants the device driver to do.

The strategy routine does not perform the request but rather queues the request or saves a pointer to the request header. The second entry point is the interrupt routine and is called by DOS immediately after the strategy routine returns. The interrupt routine is called with no parameters. Its function is to perform the operation based on the queued request and set up any return information.

DOS passes the pointer to the request header in ES:BX. This structure consists of a fixed length header (Request Header) followed by data pertinent to the operation to be performed.

Note: It is the responsibility of the device driver to preserve the machine state. For example, save all registers on entry, and restore them on exit.

The stack used by DOS has enough room on it to save all of the registers. If more stack space is needed, it is the device driver's responsibility to allocate and maintain another stack.

All calls to device drivers are FAR calls. FAR returns should be executed to return to DOS.

Installing Character Devices

One of the functions defined for each device is INIT. This routine is called only once when the device is installed and never again. The INIT routine returns the following:

- A location to the first free byte of memory after the device driver, like a terminate and stay resident that is stored in the ending address field. This way, the initialization code can be used once and thrown away to save space.
- After setting the ending address field, a character device driver can set the status word and return.

Installing Block Devices

Block devices are installed in the same way character devices are. The difference is that block devices return additional information. Block devices must also return:

- The number of units for the block device. This number determines the logical names that the devices will have. For example, if the current maximum logical device letter is F at the time of the install call, and the block device driver INIT 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 on the device. The number of units returned by INIT overrides the value in the name/unit field of the device header.
- A pointer to a BPB (BIOS parameter block) pointer array. This is a pointer to an array of n word pointers where n is the number of units defined. These word pointers point to BPB's. This way, if all of the units are the same, the entire array can point to the same BPB to save space.

The BPB contains information pertinent to the devices such as the sector size, the number of sectors per allocation unit, and so forth. The sector size in the BPB cannot be greater than the maximum allotted size set at DOS initialization time.

Note: This array must be protected below the free pointer set by the return.

- The media descriptor byte. This byte is passed to devices so that they know what parameters DOS is currently using for a particular drive unit.

Block devices can take several approaches. They can be *dumb* or *smart*. A dumb device would define a unit (and therefore a BPB) for each possible media drive combination. Unit 0 = drive 0; single side, unit 1 = drive 0; double side, etc. For this approach, media descriptor bytes would mean nothing. A smart device would allow multiple media per unit. In this case, the BPB table returned at INIT must define space large enough to accommodate the largest possible media supported (sector size in BPB must be as large as maximum sector size that DOS is currently using). Smart drivers will use the “media byte” to pass information about what medium is currently in a unit.

Request Header

The request header passes the information describing what DOS wants the device driver to do.

Field	Length
Length in bytes of the request header plus any data at the end of the request header.	BYTE
Unit code. The subunit the operation is for (minor device). Has no meaning for character devices.	BYTE
Command code.	BYTE
Status.	WORD
Area reserved for DOS.	8-BYTE
Data appropriate to the operation.	Variable

Unit Code Field

The unit code field identifies which unit in a block device driver the request is for. For example, if a block device driver has three units defined, then the possible values of the unit code field would be 0, 1, and 2.

Command Code Field

The command code field in the request header can have the following values:

Code	Function
0	INIT
1	MEDIA CHECK (Block only, NOP for character)
2	BUILD BPB (Block only, NOP for character)
3	IOCTL input (only called if IOCTL bit is 1)
4	INPUT (read)
5	NONDESTRUCTIVE INPUT NO WAIT (Character devices only)
6	INPUT STATUS (Character devices only)
7	INPUT FLUSH (Character devices only)
8	OUTPUT (write)
9	OUTPUT (write) with verify
10	OUTPUT STATUS (Character devices only)
11	OUTPUT FLUSH (Character devices only)
12	IOCTL output (only called if IOCTL bit is 1)
13	DEVICE OPEN (only called if OPEN/CLOSE/RM bit is set)
14	DEVICE CLOSE (only called if OPEN/CLOSE/RM bit is set)
15	REMOVABLE MEDIA (only called if OPEN/CLOSE/RM bit is set and device type is block)

Note: Command codes 13, 14, and 15 are for use with DOS versions 3.00 and 3.10.

Status Field

The status field in the request header contains:

15	14-10	9	8	7-0
E R R O R	RESERVED	B U S Y	D O N E	ERROR CODE (bit 15 on)

The status word field is zero on entry and is set by the driver interrupt routine on return.

Bit 15 is the error bit. If this bit is set, the low 8 bits of the status word (7-0) indicate the error code.

Bits 14 - 10 are reserved.

Bit 9 is the busy bit. It is only set by status calls and the removable media call. See "STATUS" and "REMOVABLE MEDIA" in this chapter for more information about the calls.

Bit 8 is the done bit. If it is set, it means the operation is complete. The driver sets the done bit to 1 when it exits.

Bits 7–0 are the low 8 bits of the status word. If bit 15 is set, bits 7–0 contain the error code. The error codes and errors are:

Error Codes	Description
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

Device Driver Functions

All strategy routines are called with ES:BX pointing to the request header. The interrupt routines get the pointers to the request header from the queue the strategy routines store them in. The command code in the request header tells the driver which function to perform.

Note: All DWORD pointers are stored offset first, then segment.

The following function call parameters are described:

- INIT
- MEDIA CHECK
- BUILD BPB (BIOS Parameter Block)
- MEDIA DESCRIPTOR BYTE
- INPUT or OUTPUT
- NONDESTRUCTIVE INPUT NO WAIT
- STATUS
- FLUSH
- OPEN or CLOSE
- REMOVABLE MEDIA

INIT

Command code=0

ES:BX

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
For DOS version 3.10, this field contains the drive number	BYTE

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, etc.).
- 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 INIT time, the BPB pointer field points to a buffer containing the information passed in CONFIG.SYS following the =. The buffer that DOS passes to the driver at INIT after the file specification contains an ASCII string for the file OPEN. The ASCII string (ending in 0H) is terminated by a carriage return (0DH) and linefeed (0AH). If there is no parameter information after the file specification, the file specification is immediately followed by a linefeed (0AH). This information is read-only and only system calls 01H-0CH and 30H can be issued by the INIT 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 etc.

If an INIT routine determines that it cannot set up the device and wants to abort without using any memory, follow this 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).

Note: If there are multiple device drivers in a single memory image file, the ending address returned by the last INIT called is the one DOS uses. It is recommended that all device drivers in a single memory image file return the same ending address.

MEDIA CHECK

Command code=1

ES:BX

Field	Length
Request header	13-BYTE
Media descriptor from DOS	BYTE
Return	BYTE
If you are using DOS 3.00 or 3.10, this call returns 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 calls MEDIA CHECK for a drive unit and passes its current Media Descriptor byte. See “Media Descriptor Byte” later in this chapter for more information about the byte. MEDIA CHECK returns one of the following:

- Media Not Changed
- Media Changed
- Not Sure
- Error code

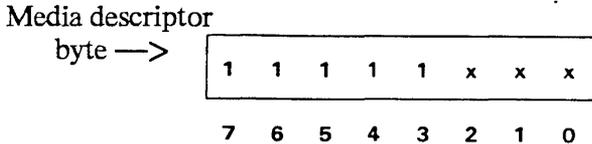
The driver must perform the following:

- Set the status word in the request header.
- Set the return byte:
 - 1 Media has been changed
 - 0 Don't know if media has been changed
 - 1 Media has not been changed

DOS 3.00 and 3.10: If the driver has set the removable media bit 11 of the device header attribute word to 1 and the driver returns -1 (media changed), the driver must set the DWORD pointer to the previous volume identification field. If DOS determines that the media changed is an error, DOS generates an error 0FH (Invalid Disk Change) on behalf of the device. If the driver does not implement volume identification support, but has bit 11 set to 1, the driver should set a pointer to the string "NO NAME ", 0.

Media Descriptor Byte

Currently the media descriptor byte has been defined for a few 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:



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	

Examples of current DOS media descriptor bytes:

Disk Type	# Sides	# Sectors/Track	Media Descriptor
Fixed disk	--	--	F8H
5 1/4-in.	2	15	F9H
5 1/4-in.	1	9	FCH
5 1/4-in.	2	9	FDH
5 1/4-in.	1	8	FEH
5 1/4-in.	2	8	FFH
8-in.	1	26	FEH
8-in.	2	26	FDH
8-in.	2	8	FEH

Note: The two MEDIA descriptor bytes that are the same for 8-in. diskettes (FEH) is not a misprint. To determine whether you are using a single sided or a double sided diskette, attempt to read the second side, and if an error occurs you can assume the diskette is single sided.

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 (BIOS Parameter Block)

Command code=2

ES:BX

Field	Length
Request header	13-BYTE
Media descriptor from DOS	BYTE
Transfer address (buffer address)	DWORD
Pointer to BPB table	DWORD

DOS calls BUILD BPB 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 that has not yet been written to the disk.

The driver must perform the following:

- Set the pointer to the BPB.
- Set the status word in the request header.

The driver must determine the correct media that is currently in the unit to return the pointer to the BPB table. The way the buffer is used (pointer passed by DOS) is determined by the non-IBM format bit in the attribute field of the device header. If bit 13 = 0 (device is IBM format compatible), the buffer contains the first sector of the FAT (most importantly the FAT ID byte). The driver must not alter this buffer in this case. If bit 13 = 1, the buffer is a one sector scratch area that can be used for anything.

For drivers that support volume identification and disk change, this call should cause a new volume identification to be read off the disk. This call indicates that the disk has legally changed.

If the device is IBM format compatible, it must be true that the first sector of the first FAT is located at the same sector for all possible media. This is because the FAT sector is read *before* the media is actually determined.

The information relating to the BPB for a particular media is kept in the boot sector for the media. In particular, the format of the boot sector is:

For DOS 2.10, 3 BYTE near JUMP (E9H) or for DOS 3.00 and 3.10, 2 BYTE short JUMP (EBH) followed by a NOP (90H)
8 BYTES OEM name and version
WORD bytes per sector
BYTE sectors per allocation unit (must be a power of 2)
WORD reserved sectors (starting at logical sector 0)
BYTE number of FATs
WORD number of root dir entries (maximum allowed)
WORD number of sectors in logical image (total sectors in media, including boot sector, directories, etc.)
BYTE media descriptor
WORD number of sectors occupied by a single FAT
WORD sectors per track
WORD number of heads
WORD number of hidden sectors

The three words at the end are intended to help the device driver understand the media. The number of heads is useful for supporting different multihead drives that have the same storage capacity but a different number of surfaces. The number of hidden sectors is useful for supporting drive partitioning schemes.

INPUT or OUTPUT

Command codes=3,4,8,9, and 12

ES:BX

Field	Length
Request header	13-BYTE
Media descriptor byte	BYTE
Transfer address (buffer address)	DWORD
Byte/sector count	WORD
Starting sector number (no meaning on character devices)	WORD
For DOS 3.00 and 3.10, pointer to the volume identification if error code 0FH is returned	DWORD

The driver must perform the following:

- Set the status word in the request header.
- Perform the requested function.
- Set the actual number of sectors (or bytes) transferred.

Note: No error checking is performed on an IOCTL I/O call. However, the driver must set the return sector (byte) count to the actual number of bytes transferred.

The following applies to block device drivers: Under certain circumstances the device driver may be asked to do a write operation of 64K bytes that seems to be a *wrap around* of the transfer address in the device driver request packet. This arises due to an optimization added to the write code in DOS. It will only happen on WRITES that are within a sector size of 64K bytes on files that are being extended past the current end of file. It is allowable for the device driver to ignore the balance of the WRITE that wraps around, if it so chooses. For example, a WRITE of 10000H bytes worth of sectors with a transfer address of XXXX:1, ignores the last two bytes.

Remember: A program that uses DOS function calls can never request an input or output operation of more than FFFFH bytes; therefore, a wrap around in the transfer (buffer) segment cannot occur. It is for this reason that 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 put a DWORD pointer to an ASCIIZ string which is the correct volume identification to ask the user to reinsert the disk.

DOS 3.00 and 3.10: 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 all right, 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.

NONDESTRUCTIVE INPUT NO WAIT

Command code=5

ES:BX

Field	Length
Request header	13-BYTE
Read from device	BYTE

The driver must perform the following:

- Return a byte from the device.
- Set the status word in the request header.

If the character device returns busy bit = 0 (characters in buffer), then the next character that would be read is returned. This character is not removed from the input buffer (hence the term *nondestructive input*). This call allows DOS to look ahead one input character.

STATUS

Command codes=6 and 10

ES:BX

Field	Length
Request header	13-BYTE

The driver must perform the following:

- Perform the requested function.
- Set the busy bit.
- Set the status word in the request header.

The busy bit is set as follows:

For 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, there is no current request. Therefore, a write request would start immediately.

For 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, there are characters in the device buffer and a read returns quickly. It also indicates that the user has typed something. DOS 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 does not hang waiting for information to be put in a buffer that does not exist.

FLUSH

Command codes=7 and 11

ES:BX

Field	Length
Request header	13-BYTE

This call tells the driver to flush (terminate) all pending requests that it has knowledge of. Its primary use is to flush the input queue on character devices.

The driver must:

Set status word in the Request Header upon return.

OPEN or CLOSE (DOS 3.00 and 3.10)

Command codes=13 and 14

ES:BX

Field	Length
Static 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 causes the device to increment the reference count. Every CLOSE causes the device to decrement the reference count. When the reference count is 0, it means there are no open files on the device. Therefore, the device should flush buffers inside the device that it has written to because now the user can change the media on a removable media drive. If the media has been changed, it is advisable to reset the reference count to 0 without flushing the buffers. This can be thought of as "last close causes flush." These calls are more useful on character devices. The OPEN call can be used to send a device initialization string. On a printer, this could cause a string to be sent that would set the the font, the page size, etc., so that the printer would always be in a known state at the start of an I/O stream. Similarly the CLOSE call can be used to send a post string (like a form feed) at the end of an I/O stream. Using IOCTL to set these pre and post strings provides a flexible mechanism of serial I/O device stream control.

Note: Since all processes have access to STDIN, STDOUT, STDERR, STDAUX, and STDPRN (handles 0,1,2,3,4), the CON, AUX, and PRN devices are always open.

REMOVABLE MEDIA (DOS 3.00 and 3.10)

Command code=15

ES:BX

Field	Length
Static request header	13-BYTE

To use this call, set bit 11 of the attribute field to 1. Block devices can only use this call through a subfunction of the IOCTL function call (44H). This call is useful because it allows a utility to know whether it is dealing with a nonremovable media drive or with a removable media drive. For example, the FORMAT utility needs to know whether a drive is removable or nonremovable because it prints 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.

Note: No error bit checking is performed. It is assumed that this call always succeeds.

The CLOCK\$ Device

A popular feature is a "Real Time Clock" board. To allow this board to be integrated into the system for TIME and DATE, there is a special device (determined by the attribute word) which is the CLOCK\$ device. This device defines and performs functions like any other character device (most functions will be set done bit, reset error bit, return). When a read or write to this device occurs, exactly 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 1/100 is seconds; and the sixth is seconds. Reading the CLOCK\$ device gets the date and time, writing to it sets the date and time.

Sample Device Driver

The DOS 3.00 and 3.10 Supplemental diskettes contain a sample device driver listing called VDISK.LST. Use the PRINT command to print a copy of the listing for reference.

Chapter 3. Using Extended Screen and Keyboard Control

Contents

Introduction	3-3
Control Sequences	3-3
Control Sequence Syntax	3-4
Cursor Control Sequences	3-6
Cursor Position	3-6
Cursor Up	3-7
Cursor Down	3-7
Cursor Forward	3-8
Cursor Backward	3-8
Horizontal and Vertical Position	3-9
Cursor Position Report	3-10
Device Status Report	3-10
Save Cursor Position	3-12
Restore Cursor Position	3-12
Erasing	3-13
Erase in Display	3-13
Erase in Line	3-13
Mode of Operation	3-14
Keyboard Key Reassignment	3-17

Introduction

This chapter explains how you can issue special control character sequences to:

- Control the position of the cursor
- Erase text from the screen
- Set the mode of operation
- Redefine the meaning of keyboard keys

Control Sequences

The control sequences are valid if you issue them through DOS function calls that use standard input, standard output, or standard error output devices. These are the function calls 01H, 02H, 06H, 07H, 09H, 0AH and 40H.

The extended screen and keyboard control device driver ANSI.SYS must be installed by placing the following statement in the configuration file CONFIG.SYS:

```
device=[d:][path]ansi.sys
```

The size of DOS in memory increases by the size of ANSI.SYS.

Control Sequence Syntax

Each of the cursor control sequences is in the format:

ESC [*parameters* COMMAND

ESC	The 1-byte ASCII code for ESC (1BH). It is not the three characters ESC.
[The character [.
<i>parameters</i>	The numeric values you specify for #. The # represents a numeric parameter. A numeric parameter is an integer value specified with ASCII characters. If you do not specify a parameter value, or if you specify a value of 0, the default value for the parameter is used.
COMMAND	An alphabetic string that represents the command. It is case specific.

For example:

```
ESC [2;10H
```

could be created using BASIC as follows:

```
The IBM Personal Computer Basic  
Version 3.00 Copyright IBM Corp. 1981, 1982, 1983, 1984  
xxxxx Bytes free
```

```
Ok  
open "sample" for output as 1  
Ok  
print #1, CHR$(27);"[2;10H";"x row 2 col 10"  
Ok  
close #1  
Ok
```

Notice that "CHR\$(27)" is ESC.

Cursor Control Sequences

The following tables contain the cursor control sequences you can use to control cursor positioning.

Cursor Position

Cursor Position	Function
ESC [#;#H	Moves the cursor to the position specified by the parameters. The first parameter specifies the row number and the second parameter specifies the column number. The default value is 1. If no parameter is given, the cursor is moved to the home position.

This example copies the file SAMPLE from the previous example, to CON, which places the cursor on row 2 column 10 of the screen:

```
type sample
```

Cursor Up

Cursor Up	Function
ESC [#A	Moves the cursor up one or more rows without changing the column position. The value of # determines the number of lines moved. The default value for # is 1. This sequence is ignored if the cursor is already on the top line.

Cursor Down

Cursor Down	Function
ESC [#B	Moves the cursor down one or more rows without changing the column position. The value of # determines the number of lines moved. The default value for # is 1. The sequence is ignored if the cursor is already on the bottom line.

Cursor Forward

Cursor Forward	Function
ESC [#C	Moves the cursor forward one or more columns without changing the row position. The value of # determines the number of columns moved. The default value for # is 1. This sequence is ignored if the cursor is already in the rightmost column.

Cursor Backward

Cursor Backward	Function
ESC [#D	Moves the cursor back one or more columns without changing the row position. The value of # determines the number of columns moved. The default value for # is 1. This sequence is ignored if the cursor is already in the leftmost column.

Horizontal and Vertical Position

Horizontal and Vertical Position	Function
ESC [#;#f	Moves the cursor to the position specified by the parameters. The first parameter specifies the line number and the second parameter specifies the column number. The default value is 1. If no parameter is given, the cursor is moved to the home position.

Cursor Position Report

Cursor Position Report	Function
ESC [#;#R	The cursor sequence report reports the current cursor position through the standard input device. The first parameter specifies the current line and the second parameter specifies the current column.

Device Status Report

Device Status Report	Function
ESC [6n	The console driver outputs a cursor position report sequence on receipt of device status report.

This example tells ANSI.SYS to put the current cursor position (row and column) in the keyboard buffer. Then ANSI.SYS reads it from the keyboard buffer and displays it on the screen.

```

PROGRAM dsr(INPUT,OUTPUT);

VAR
  f:FILE OF CHAR;
  key:CHAR;

FUNCTION inkey:CHAR;           { read character  }
VAR                             { from the          }
  ch:CHAR;                       { keyboard buffer }
BEGIN
  READ(f,ch);
  inkey:=ch
END;

BEGIN
  ASSIGN(f,'user');
  RESET(f);
  WRITE(CHR(27),'[6n');        { issue a DSR   }
  key:=inkey;                  { read up to    }
  key:=inkey;                  { first digit   }
  key:=inkey;                  { of the row    }
  WRITE('row ',inkey,inkey,' column ');
  key:=inkey;                  { skip to column}
  WRITE(inkey,inkey)           { write column  }
END.

```

Save Cursor Position

Save Cursor Position	Function
ESC [s	The current cursor position is saved. This cursor position can be restored with the restore cursor position sequence (see below).

Restore Cursor Position

Restore Cursor Position	Function
ESC [u	Restores the cursor to the value it had when the console driver received the save cursor position sequence.

Erasing

The following tables contain the control sequences you can use to erase text from the screen.

Erase in Display

Erase in Display	Function
ESC [2J	Erases all of the screen and the cursor goes to the home position.

Erase in Line

Erase in Line	Function
ESC [K	Erases from the cursor to the end of the line and includes the cursor position.

Mode of Operation

The following tables contain the control sequences you can use to set the mode of operation.

They are:

- Set Graphics Rendition (SGR)
- Set Mode (SM)
- Reset Mode (RM)

Set Graphics Rendition (SGR)																																															
SGR	Function																																														
ESC [#;...;#m	Sets the character attribute specified by the parameters. All following characters have the attribute according to the parameters until the next occurrence of SGR.																																														
	<table border="0"> <thead> <tr> <th>Parameter</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>All attributes off (normal white on black)</td> </tr> <tr> <td>1</td> <td>Bold on (high intensity)</td> </tr> <tr> <td>4</td> <td>Underscore on (IBM Monochrome Display only)</td> </tr> <tr> <td>5</td> <td>Blink on</td> </tr> <tr> <td>7</td> <td>Reverse video on</td> </tr> <tr> <td>8</td> <td>Canceled on (invisible)</td> </tr> <tr> <td>30</td> <td>Black foreground</td> </tr> <tr> <td>31</td> <td>Red foreground</td> </tr> <tr> <td>32</td> <td>Green foreground</td> </tr> <tr> <td>33</td> <td>Yellow foreground</td> </tr> <tr> <td>34</td> <td>Blue foreground</td> </tr> <tr> <td>35</td> <td>Magenta foreground</td> </tr> <tr> <td>36</td> <td>Cyan foreground</td> </tr> <tr> <td>37</td> <td>White foreground</td> </tr> <tr> <td>40</td> <td>Black background</td> </tr> <tr> <td>41</td> <td>Red background</td> </tr> <tr> <td>42</td> <td>Green background</td> </tr> <tr> <td>43</td> <td>Yellow background</td> </tr> <tr> <td>44</td> <td>Blue background</td> </tr> <tr> <td>45</td> <td>Magenta background</td> </tr> <tr> <td>46</td> <td>Cyan background</td> </tr> <tr> <td>47</td> <td>White background</td> </tr> </tbody> </table>	Parameter	Meaning	0	All attributes off (normal white on black)	1	Bold on (high intensity)	4	Underscore on (IBM Monochrome Display only)	5	Blink on	7	Reverse video on	8	Canceled on (invisible)	30	Black foreground	31	Red foreground	32	Green foreground	33	Yellow foreground	34	Blue foreground	35	Magenta foreground	36	Cyan foreground	37	White foreground	40	Black background	41	Red background	42	Green background	43	Yellow background	44	Blue background	45	Magenta background	46	Cyan background	47	White background
Parameter	Meaning																																														
0	All attributes off (normal white on black)																																														
1	Bold on (high intensity)																																														
4	Underscore on (IBM Monochrome Display only)																																														
5	Blink on																																														
7	Reverse video on																																														
8	Canceled on (invisible)																																														
30	Black foreground																																														
31	Red foreground																																														
32	Green foreground																																														
33	Yellow foreground																																														
34	Blue foreground																																														
35	Magenta foreground																																														
36	Cyan foreground																																														
37	White foreground																																														
40	Black background																																														
41	Red background																																														
42	Green background																																														
43	Yellow background																																														
44	Blue background																																														
45	Magenta background																																														
46	Cyan background																																														
47	White background																																														

Set Mode (SM)																			
SM	Function																		
ESC [=#h or ESC [=h or ESC [=0h or ESC [?7h	Invokes the screen width or type specified by the parameter. <table> <thead> <tr> <th>Parameter</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>40x25 black and white</td> </tr> <tr> <td>1</td> <td>40x25 color</td> </tr> <tr> <td>2</td> <td>80x25 black and white</td> </tr> <tr> <td>3</td> <td>80x25 color</td> </tr> <tr> <td>4</td> <td>320x200 color</td> </tr> <tr> <td>5</td> <td>320x200 black and white</td> </tr> <tr> <td>6</td> <td>640x200 black and white</td> </tr> <tr> <td>7</td> <td>Wrap at end of line. (Typing past end-of-line results in new line.)</td> </tr> </tbody> </table>	Parameter	Meaning	0	40x25 black and white	1	40x25 color	2	80x25 black and white	3	80x25 color	4	320x200 color	5	320x200 black and white	6	640x200 black and white	7	Wrap at end of line. (Typing past end-of-line results in new line.)
Parameter	Meaning																		
0	40x25 black and white																		
1	40x25 color																		
2	80x25 black and white																		
3	80x25 color																		
4	320x200 color																		
5	320x200 black and white																		
6	640x200 black and white																		
7	Wrap at end of line. (Typing past end-of-line results in new line.)																		

Reset Mode (RM)	
RM	Function
ESC [=#l or ESC [=l or ESC [=0l or ESC [?7l	Parameters are the same as SM (Set Mode) except that parameter 7 resets wrap at end-of-line mode (characters past end-of-line are thrown away).

Keyboard Key Reassignment

The following table contains the control sequences you can use to redefine the meaning of keyboard keys.

The control sequence is:	Function
ESC [#;#;...#p or ESC ["string"p or ESC [#;"string";#; #;"string";#p or any other combination of strings and decimal numbers	The first ASCII code in the control sequence defines which code is being mapped. The remaining numbers define the sequence of ASCII codes generated when this key is intercepted. However, if the first code in the sequence is 0 (NULL) the first and second code make up an extended ASCII redefinition (see Chapter 6 for a list of extended ASCII codes).

Here are some examples:

To execute these examples, you can either:

- Create a file that contains the following statements and then use the TYPE command to display the file that contains the statement.
 - Execute the command at the DOS prompt.
1. Reassign the Q and q key to the A and a (and the other way as well):

Creating a File:

```
ESC [65;81p    A becomes Q
ESC [97;113p   a becomes q
ESC [81;65p    Q becomes A
ESC [113;97p   q becomes a
```

At the DOS Prompt:

```
prompt $e[65;81p    A becomes Q
prompt $e[97;113p   a becomes q
prompt $e[81;65p    Q becomes A
prompt $e[113;97p   q becomes a
```

2. Reassign the F10 key to a DIR command followed by a carriage return:

Creating a File:

```
ESC [0;68;"dir";13p
```

At the DOS Prompt:

```
prompt $e[0;68;"dir";13p
```

The \$e is the prompt command characters for ESC. The 0;68 is the extended ASCII code for the F10 key; 13 decimal is a carriage return.

- The following example sets the prompt to display the current directory on the top of the screen and the current drive on the current line.

```
prompt $e[s$e[1;30f$e[K$pe[u$n$g
```

If the current directory is C:\FILES, and the current drive is C, this example would display:

C:\FILES

C>

- The following assembly language program reassigns the F10 key to a DIR B: command followed by a carriage return.

```
TITLE SETANSI.ASM - SET F10 TO STRING FOR ANSI.SYS
CSEG      SEGMENT PARA PUBLIC 'CODE'
          ASSUME CS:CSEG,DS:CSEG

          ORG      100H      -
ENTPT:    JMP      SHORT START
STRING    DB       27, '[0;68:"DIR B: ";13p' ;REDEFINE F10 KEY
STRSIZ    EQU     $-STRING ;LENGTH OF ABOVE MESSAGE
HANDLE    EQU     1         ;PRE-DEFINED FILE
          ;HANDLE FOR STANDARD OUTPUT

START     PROC     NEAR
          MOV     BX,HANDLE ;STANDARD OUTPUT DEVICE
          MOV     CX,STRSIZ ;GET SIZE OF TEXT TO BE SENT
          MOV     DX,OFFSET STRING ;PASS OFFSET OF STRING
          ;TO BE SENT
          MOV     AH,40H    ;FUNCTION="WRITE TO DEVICE"
          INT     21H      ;CALL DOS
          RET          ;RETURN TO DOS
START     ENDP

CSEG      ENDS
          END      ENTPT
```


Chapter 4. File Management Notes

Contents

Introduction	4-3
Version Specific Information	4-3
File Management Functions	4-3
FCB Function Calls	4-5
Handle Function Calls	4-6
Special File Handles	4-8
ASCII and Binary Mode	4-9
File I/O in Binary Mode	4-10
File I/O in ASCII Mode	4-11
Number of Open Files Allowed	4-12
Restrictions on FCB Usage	4-12
Restrictions on Handle Usage	4-13
Allocating Space to a File	4-14

Introduction

This chapter tells you how to:

- Use file management functions (FCB function calls and Handle function calls)
- Do file I/O in ASCII mode and Binary mode

Version Specific Information

The following information in this chapter is specific to a version of DOS:

Restrictions on FCB Usage: For DOS 3.00 and 3.10, the number of files opened using FCBs is limited if SHARE is loaded and if the FCBS command is specified.

File Management Functions

Use DOS function calls to create, open, close, read, write, rename, find, and erase files. There are two sets of function calls that DOS provides for support of file management. They are:

- File Control Block function calls (FCB function calls 0FH – 24H)
- Extended function calls (Handle function calls 39H – 62H)

Handle function calls are easier to use and more powerful than FCB function calls. The following table compares the use of FCB function calls to Handle function calls.

FCB Calls	Handle Calls
Addresses files that are only in the current directory.	Addresses files in <i>any</i> directory.
Requires that the application program maintain a file control block to open, create, rename, or delete a file. For I/O requests, the application program also needs an FCB.	Does not require maintenance of an FCB. Requires a string that contains the drive, path, and filename to open, create, rename, or delete a file. For file I/O requests, the application program only has to maintain a 16-bit word (file handle) that is supplied by DOS.

The only reason an application should use FCB function calls is to maintain the ability to run under DOS version 1.10. To do this, a program can only use function calls supplied by DOS 1.10 (00H - 2EH).

FCB Function Calls

FCB function calls require the use of one file control block per open file, which is maintained by the application program and DOS. The application program supplies a pointer to the FCB and fills in the appropriate fields required by the specific function call. An FCB function call can perform file management on any valid drive on the system, but only in the current directory of the specified drive. By using the current block, current record, and record length fields of the FCB, you can perform sequential I/O by using the sequential read or write function calls. Random I/O can be performed by filling in the random record and record length fields. See "File Control Block" on page 7-12 for information on the FCB structure.

Several possible uses of FCB type calls are considered programming errors and should not be done under any circumstances. This is to avoid problems with file sharing and compatibility. One such error occurs when a program uses the same FCB structure to access more than one open file. By opening a file using an FCB, doing I/O, and then replacing the filename field in the file control block with a new filename, a program can then open a second file using the same FCB. This is invalid because DOS writes control information about the file into the reserved fields of the FCB. This information is changed when the second file is opened using the same FCB. If the program then replaces the filename field with the original filename and then tries to perform I/O to this file, DOS may become confused because the control information has been changed. An FCB should never be used to open a second file without closing the file that is currently open. If more than one file is to be open concurrently, separate FCBs should be used.

A program should also never tamper with the DOS reserved fields in the FCB, as the contents and structure of these fields change in different versions of DOS. It is also good programming practice to close all files after all I/O to a file is done. This avoids potential file sharing problems that require a limit on the number of files concurrently open using FCB function calls. A delete or a rename on a file that is currently open is also considered an error and should not be attempted by an application program.

Handle Function Calls

The recommended method of file management is by using the extended "handle" set of function calls. These calls are not restricted to files in the current directory. Also, the handle set of file management calls allow the application program to define the type of access that other processes can have concurrently with the same file if file sharing is loaded.

To create or open a file, the application supplies a pointer to an ASCIIZ string giving the name and location of the file. An ASCIIZ string contains an optional drive letter, optional path and mandatory file specification, terminated by a byte of 00H. The following is an example of an ASCIIZ string:

```
DB "a:\path\filename.ext",0
```

If the file is being created, the application program also supplies the attribute of the file. This is a set of values that defines if the file is read only, hidden, system, directory, or volume label. See "DOS Disk Directory" on page 5-10 for information on file attributes.

If the file is being opened, the program can define the sharing and access modes that the file is opened in. The access mode informs DOS what operations your program will perform on this file (read-only, write-only or read/write). The sharing mode controls the type of operations other processes may perform concurrently on the file. A program can also control if a child process inherits the open files of the parent. The sharing mode field has meaning only if file sharing is loaded when the file is opened.

To rename or delete a file, the application program simply needs to provide a pointer to the ASCIIZ string containing the name and location of the file and another string with the new name if the file is being renamed.

The open or create function calls return a 16-bit value referred to as the file handle. To do any I/O to a file, the program uses this handle to reference the file. Once a file is opened, a program no longer needs to maintain the ASCIIZ string pointing to the file, nor is there any requirement to stay in the same directory. DOS keeps track of the location of the file regardless of what directory is current.

Sequential I/O can be performed using the handle read (3FH) or write (40H) function calls. The offset in the file that I/O is performed to is automatically moved to the end of what was just read or written. If random I/O is desired, the LSEEK (42H) function call can be used to set the offset into the file that the I/O is performed at.

Special File Handles

DOS sets up five special file handles for use by application programs. These 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)

These handles are predefined by DOS and can be used by an application program. They do not need to be opened by the program, although a program can close these handles. Stdin should be treated as a read-only file, and Stdout and Stderr should be treated as write only handles. Stdin and Stdout can be redirected. All handles inherited by a process can be redirected, but not at the command line.

These handles are very useful for doing I/O to and from the console device. For example, you could read input from the keyboard using the read (3FH) function call and file handle 0000H (Stdin), and write output to the console screen with the write function call (40H) and file handle 0001H (Stdout). If you wanted an output that could not be redirected, you could output it using file handle 0002H (Stderr). This is very useful for error messages or prompts that a user must see in order to act upon them.

File handles 0003H (Stdaux) and 0004H (Stdprn) can both be read from and written to. Stdaux is typically a serial device and stdprn is usually a parallel device.

ASCII and Binary Mode

I/O to files is done in binary mode. This means that the data is read to or written from a file without modification. However, DOS can also read or write to devices in ASCII mode. In ASCII mode, DOS does some string processing and modification to the characters read or written. The predefined handles are in ASCII mode when initialized by DOS. All other file handles that don't refer to devices are in binary mode. A program can use the IOCTL (44H) function call to set the mode that I/O is done to a device. The predefined file handles are all devices, so the mode can be changed from ASCII to binary via IOCTL. Regular file handles that are not devices are always in binary mode, and they cannot be changed to ASCII mode.

The predefined handles Stdin (0000H), Stdout (0001H), and Stderr (0002H) are all duplicate handles. If the IOCTL function call is used to change the mode of any of these three handles, the mode of all three handles is changed. For example, if IOCTL was used to change Stdout to binary mode, then Stdin and Stderr would also be changed to binary mode.

File I/O in Binary Mode

When a file is read in binary mode:

- The characters $\wedge S$ (Scroll lock), $\wedge P$ (Print Screen), $\wedge C$ (Control Break) are not checked for during the read. Therefore, no printer echo occurs if $\wedge S$ or $\wedge P$ are read.
- There is no echo to Stdout (0001H).
- Reads the number of specified bytes and returns immediately when the last byte is received or the end of file is reached.
- Allows no editing of the line input using the function keys if the input is from Stdin (0000H).

When a file is written in binary mode:

- The characters $\wedge S, \wedge P, \wedge C$ are not checked for during the write operation. Therefore there is no printer echo.
- There is no echo to Stdout (0001H).
- The exact number of bytes specified are written.
- Does not caret control characters. For example, control D is sent out as byte 04H instead of the two bytes \wedge and D.
- Does not expand tabs into spaces.

File I/O in ASCII Mode

When a file is read in ASCII mode:

- Checks for the characters ^C, ^S, and ^P.
- Returns as many characters as there are in the device input buffer, or the number of characters requested, whichever is less. If the number of characters requested was less than the number of characters in the device input buffer, then the next read will address the remaining characters in the buffer.
- If there are no more bytes remaining in the device input buffer, read a line (terminated with ^M) into the buffer. This line may be edited with the function keys. The characters returned terminate with a sequence of 0DH,0AH (^M,^J) if the number of characters requested is sufficient to include them. For example, if 5 characters were requested, and only 3 were entered before the carriage return (0DH or ^M) was presented to DOS from the console device, the 3 characters entered and 0DH, and 0AH would be returned. However, if 5 characters were requested and 7 were entered before the carriage return, only the first 5 characters would be returned. No 0DH,0AH sequence would be returned in this case. If less than the number of characters requested are entered when the carriage return is received, the characters received and the 0DH,0AH would be returned. The reason the 0AH (line feed or ^J) byte is added to the returned characters is to make devices look like text files.
- If a 1AH (^Z) is found, the input is terminated at that point. No 0DH,0AH sequence is added to the string.
- Echoing is performed.

- Tabs are expanded into spaces on echo. They are left as a tab byte (09H) in the input buffer.

When a file is written in ASCII mode:

- The characters ^S, ^P, and ^C are checked for during the write operation.
- Expands tabs to 8-character boundaries and fills with spaces (20H).
- Carets control characters. For example, ^D is written as two bytes, ^ and D.
- Bytes are output until the number specified is output or until a ^Z is found. The number actually output is returned to the user.

Number of Open Files Allowed

The number of files that can be open concurrently is restricted by DOS. This number is determined by how the file is opened or created (FCB or handle function call) and the number specified by the FCBS and FILES commands in the CONFIG.SYS file. The number of files allowed open by FCB function calls and the number of files that can be opened by handle type calls are independent of one another.

Restrictions on FCB Usage

If file sharing is not loaded using the SHARE command, there are no restrictions on the number of files concurrently open using FCB function calls. However, when file sharing is loaded, the maximum number of FCB opened files is limited by the value set by the FCBS command in the CONFIG.SYS configuration file. For information on the FCBS

command, refer to Chapter 4 of the *DOS Reference* for versions 3.00 and 3.10. The FCBS command has two values that you can specify m , n . The value for m specifies the total number of files that can be opened by FCBS, and the value for n specifies the number of files opened by FCBS that are protected from being closed.

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

When DOS determines the least recently used file to close, it does not include the first n files opened, therefore the first n are protected from being closed.

Restrictions on Handle Usage

The number of file handles that can be open at one time by all processes is determined by the FILES command in the CONFIG.SYS file (for more information see the FILES command in the *DOS Reference*). The number of files a single process can open depends on the value specified for FILES command. If FILES is greater than or equal to 20, a single process can open 20 files. If the value specified for FILES is less than 20, a single process can open less than 20 files. This value includes three predefined handles. One handle is for standard input/output/error, one for standard auxiliary, and one for standard printer. This means a single process can open a maximum of 17 additional handles (20 minus 3).

Allocating Space to a File

Files are not necessarily written sequentially on a disk. Space is allocated as it is needed and the next location available on the disk is allocated as the next location for a file being written. Therefore, if considerable file creation and erasure activity has taken place, newly created files may not be written in sequential sectors. However, due to the mapping (chaining) of file space via the File Allocation Table (FAT), and the function calls available, any file can be used in either a sequential or random manner.

Space is allocated in increments called clusters. Cluster size varies from a low of one sector of disk space per cluster on a single-sided diskette to a higher number of sectors/cluster on other disk formats. The cluster size of a fixed disk is based on the size of the DOS partition, and is determined when the fixed disk is formatted with the FORMAT command. For example, for a 10M byte fixed disk that is totally dedicated to one DOS partition, the cluster size is equal to 8 sectors.

An application program should not concern itself with the way that DOS allocates disk space to a file. The size of a cluster is only important in that it determines the smallest amount of space allocated to a file at one time. For example, a diskette with 2 sectors per cluster and a sector size of 512 bytes would allocate diskette space to a file in 1024 byte blocks. Therefore, even if a file was less than one cluster long, a cluster's worth of disk space would be allocated to the file. If more disk space is needed, additional clusters are allocated to the file. A disk is considered full when all the available clusters have been allocated to files.

Chapter 5. DOS Disk Allocation

Contents

Introduction	5-3
Version Specific Information	5-3
The DOS Area	5-4
The Boot Record	5-4
DOS File Allocation Table (FAT)	5-5
How to Use the File Allocation Table for 12-Bit FAT Entries	5-8
How to Use the File Allocation Table for 16-Bit FAT Entries	5-9
DOS Disk Directory	5-10
Directory Entries	5-10
Bytes 0-7	5-10
Bytes 8-10	5-11
Byte 11	5-11
Bytes 12-21	5-12
Bytes 22-23	5-12
Bytes 24-25	5-13
Bytes 26-27	5-13
Bytes 28-31	5-13
The Data Area	5-14

Introduction

This chapter contains the following information about DOS:

- The boot record
- The DOS file allocation table (FAT) for 12-bit and 16-bit FATs
- The DOS disk directory
- The data area

Version Specific Information

The following information in this chapter is specific to a version of DOS:

DOS File Allocation Table (FAT):

- 12-bit FATs are for use with DOS versions 2.10, 3.00, and 3.10.
- 16-bit FATs are for use with DOS versions 3.00 and 3.10.

Also, for DOS versions 3.00 and 3.10, the File Allocation Table indicator F9H is used to identify 15 sector-per-track diskettes.

The DOS Area

All disks and diskettes formatted by DOS are created with a sector size of 512 bytes. The DOS area (entire diskette for diskettes, DOS partition for fixed disks) is formatted as follows:

Boot record - 1 sector
First copy of file allocation table (FAT) - variable size
Second copy of file allocation table - same size as first copy of FAT
Root directory - variable size
Data area

The following sections describe each of the allocated areas.

The Boot Record

The boot record resides on track 0, sector 1, side 0 of every diskette formatted by the DOS FORMAT command. It is put on all disks to produce an error message if you try to start up the system with a nonsystem diskette in drive A. For fixed disks, the boot record resides on the first sector of the DOS partition.

DOS File Allocation Table (FAT)

This section explains how DOS uses the file allocation table (FAT) to convert the clusters of a file to logical sector numbers. We recommend that system utilities use the DOS handle function calls rather than interpreting the FAT.

The FAT is used by DOS to allocate disk space for a file, one cluster at a time.

The FAT consists of a 12-bit entry (1.5 bytes) for each cluster on the disk or a 16-bit entry (2 bytes) when a fixed disk has more than 20740 sectors as is the case for fixed disks larger than 10M bytes.

The first two FAT entries map a portion of the directory; these FAT entries contain indicators of the size and format of the disk. The FAT can be in a 12-bit or a 16-bit format. DOS determines whether a disk has a 12- or 16-bit FAT by looking at the total number of allocation units on the disk. For all diskettes and fixed disks with DOS partitions less than 20740 sectors, the FAT uses a 12-bit value to map a cluster. For larger partitions, DOS uses a 16-bit value.

The second, third, and fourth (if applicable for 16-bit FATs) bytes always contain FFFFH. The first byte is used as follows:

Hex Value	Meaning
FF	Dual sided, 8 sector-per-track diskette.
FE	Single sided, 8 sector-per-track diskette.
FD	Dual sided, 9 sector-per-track diskette.
FC	Single sided, 9 sector-per-track diskette.
F9	Dual sided, 15 sector-per-track diskette.
F8	Fixed disk.

The third FAT entry begins the mapping of the data area (cluster 002).

Note: These values are provided as a reference. Therefore, programs should not make use of these values.

Each entry contains 3 hexadecimal characters, (or 4 for 16-bit FATs). () indicates the high-order four bit value in the case of the 16-bit FAT entries. They can be either:

Hex Value	Meaning
(0)000	if the cluster is unused and available, or
(F)FF8--(F)FFF	to indicate the last cluster of a file, or
(X)XXX	any other hexadecimal characters that are the cluster number of the <i>next cluster</i> in the file. The cluster number of the first cluster in the file is kept in the file's directory entry.

The values (F)FF0--(F)FF7 are used to indicate reserved clusters. (F)FF7 indicates a bad cluster if it is not part of an allocation chain. (F)FF8--(F)FFF are used as end-of-file marks.

The file allocation table always occupies the sector or sectors immediately following the boot record. If the FAT is larger than 1 sector, the sectors occupy consecutive sector numbers. Two copies of the FAT are written, one following the other, for integrity. The FAT is read into one of the DOS buffers whenever needed (open, allocate more space, etc.).

How to Use the File Allocation Table for 12-Bit FAT Entries

Obtain the *starting cluster* of the file from the directory entry.

Now, to locate each subsequent cluster of the file:

1. Multiply the cluster number just used by 1.5 (each FAT entry is 1.5 bytes long).
2. The whole part of the product is an offset into the FAT, pointing to the entry that maps the cluster just used. That entry contains the cluster number of the next cluster of the file.
3. Use a MOV instruction to move the word at the calculated FAT offset into a register.
4. If the last cluster used was an even number, keep the low-order 12 bits of the register; otherwise, keep the high-order 12 bits.
5. If the resultant 12 bits are (FF8-FFF)H, no more clusters are in the file. Otherwise, the 12 bits contain the cluster number of the next cluster in the file.

To convert the cluster to a logical sector number (relative sector, such as that used by INT 25H and 26H and by DEBUG):

1. Subtract 2 from the cluster number.
2. Multiply the result by the number of sectors per cluster.
3. Add the logical sector number of the beginning of the data area.

How to Use the File Allocation Table for 16-Bit FAT Entries

Obtain the *starting cluster* of the file from the directory entry. Now to locate each subsequent cluster of the file:

1. Multiply the cluster number used by 2 (each FAT entry is 2 bytes long).
2. Use MOV word instruction to move the word at the calculated FAT offset into a register.
3. If the resultant 16 bits are (FFF8-FFFF)H, no more clusters are in the file. Otherwise, the 16 bits contain the cluster number of the next cluster in the file.

DOS Disk Directory

The **FORMAT** command initially builds the root directory for all disks. Its location (logical sector number) and the maximum number of entries are available through the device driver interfaces.

Directory Entries

Since directories other than the root directory are actually files, there is no limit to the number of entries they may contain.

All directory entries are 32 bytes long, and are in the following format (byte offsets are in decimal). The following paragraphs describe the directory entry bytes:

Bytes 0-7

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. This is used to limit the length of directory searches, for performance reasons.

05H Indicates that the first character of the filename actually has an **E5H** character.

E5H Filename was 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 (**0000H** if the parent directory is the root directory).

Any other character is the first character of a filename.

Bytes 8-10

These bytes indicate the filename extension.

Byte 11

This byte indicates the file's attribute. The attribute byte is mapped as follows (values are in hexadecimal):

Note: Attributes 08H and 10H cannot be changed using function call 43H (CHMOD).

The system files (IBMBIO.COM and IBMDOS.COM) are marked as read-only, hidden, and system files. Files can be marked hidden when they are created. Also, the read-only, hidden, system, and archive attributes may be changed through the CHMOD function call.

- 01H Indicates that the file is marked read-only. An attempt to open the file for output using function call 3DH results in an error code being returned. This value can be used with other values below.
- 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 that 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 that the entry defines a subdirectory and is excluded from normal directory searches.
- 20H Indicates an archive bit. The bit is set on whenever the file has been written to and closed. It is used by the BACKUP and RESTORE

commands for determining whether the file was changed since it was last backed up. This bit can be used along with other attribute bits.

All other bits are reserved, and must be 0.

Bytes 12-21

This is a reserved area by DOS.

Bytes 22-23

These bytes 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 is the binary number of hours (0-23)

mm is the binary number of minutes (0-59)

xx is the binary number of two-second increments

Note: The time is stored with the least significant byte first.

Bytes 24-25

This area contains 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 is 1-12

dd is 1-31

yy is 0-119 (1980-2099)

Note: The date is stored with the least significant byte first.

Bytes 26-27

This area contains the starting cluster number of the first cluster in the file. The first cluster for data space on all fixed disks and diskettes is always cluster 002. The cluster number is stored with the least significant byte first.

Note: System programmers, see “DOS File Allocation Table (FAT)” for details about converting cluster numbers to logical sector numbers.

Bytes 28-31

This area contains 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.

The Data Area

Allocation of space for a file (in the data area) is done only when needed (it is not preallocated). The space is allocated one cluster (unit of allocation) at a time. A cluster is always one or more consecutive sector numbers, and all of the clusters for a file are “chained” together in the File Allocation Table.

The clusters are arranged on disk to minimize head movement for multisided media. All of the space on a track (or cylinder) is allocated before moving on to the next track. This is accomplished by using the sequential sector numbers on the lowest-numbered head, then all the sector numbers on the next head, and so on until all sectors on all heads of the track are used. Then, the next sector to be used will be sector 1 on head 0 of the next track.

For fixed disk, the size of the file allocation table and directory are determined when FORMAT initializes it, and are based on the size of the DOS partition.

For diskettes, the following table can be used:

# Sides	Sectors/Track	FAT Size Sectors	DIR Sectors	DIR Entries	Sectors/Cluster
1	8	1	4	64	1
2	8	1	7	112	2
1	9	2	4	64	1
2	9	2	7	112	2
2	15	7	14	224	1

Files in the data area are not necessarily written sequentially on the disk. The data area space is allocated one cluster at a time, skipping over clusters already allocated. The first free cluster found is the next cluster allocated, regardless of its physical location on the disk. This permits the most efficient utilization of disk space because clusters made available by erasing files can be allocated for new files. Refer back to the description of the DOS File Allocation Table in this chapter for more information.

Chapter 6. DOS Interrupts and Function Calls

Contents

Introduction	6-7
Version Specific Information	6-7
DOS Registers	6-9
Extended ASCII Codes	6-11
Interrupts	6-12
20H Program Terminate	6-12
21H Function Request	6-13
22H Terminate Address	6-13
23H Ctrl-Break Exit Address	6-13
24H Critical Error Handler Vector	6-14
Disk Errors	6-18
Handling of Invalid Responses (DOS 3.00 and 3.10)	6-18
Other Errors	6-19
25H Absolute Disk Read	6-22
26H Absolute Disk Write	6-23
27H Terminate but Stay Resident	6-24
28H-2EH Reserved for DOS	6-25
2FH Multiplex Interrupt	6-26
Function Codes	6-27
2FH Error Codes	6-27
Example 2FH Handler	6-30
Installing the Handler	6-31
30H-3FH Reserved for DOS	6-31
Function Calls	6-32
Listing of Function Calls	6-33
DOS Internal Stack	6-36
Error Return Information	6-36

DOS 2.10 Error Codes	6-37
Get Extended Error (DOS 3.00 and 3.10)	6-38
ASCIIZ Strings	6-44
Network Paths	6-45
Network Access Rights	6-45
File Handles	6-46
Using DOS Functions	6-47
00H Program Terminate	6-49
01H Keyboard Input	6-50
02H Display Output	6-51
03H Auxiliary Input	6-52
04H Auxiliary Output	6-53
05H Printer Output	6-54
06H Direct Console I/O	6-55
07H Direct Console Input Without Echo	6-56
08H Console Input Without Echo	6-57
09H Print String	6-58
0AH Buffered Keyboard Input	6-59
0BH Check Standard Input Status	6-60
0CH Clear Keyboard Buffer and Invoke a Keyboard Function	6-61
0DH Disk Reset	6-62
0EH Select Disk	6-63
0FH Open File	6-64
10H Close File	6-65

11H Search for First Entry	6-66
12H Search for Next Entry	6-68
13H Delete File	6-69
14H Sequential Read	6-70
15H Sequential Write	6-71
16H Create File	6-72
17H Rename File	6-73
19H Current Disk	6-74
1AH Set Disk Transfer Address	6-75
1BH Allocation Table Information	6-76
1CH Allocation Table Information for Specific Device	6-77
21H Random Read	6-78
22H Random Write	6-79
23H File Size	6-80
24H Set Relative Record Field	6-81
25H Set Interrupt Vector	6-82
26H Create New Program Segment	6-83
27H Random Block Read	6-84
28H Random Block Write	6-85
29H Parse Filename	6-86
2AH Get Date	6-88

2BH Set Date	6-89
2CH Get Time	6-90
2DH Set Time	6-91
2EH Set/Reset Verify Switch	6-92
2FH Get Disk Transfer Address (DTA)	6-93
30H Get DOS Version Number	6-94
31H Terminate Process and Remain Resident ...	6-95
33H Ctrl-Break Check	6-96
35H Get Vector	6-97
36H Get Disk Free Space	6-98
38H (DOS 2.10) Return Country Dependent Information	6-99
38H (DOS 3.00 and 3.10) Get or Set Country Dependent Information	6-101
39H Create Subdirectory (MKDIR)	6-106
3AH Remove Subdirectory (RMDIR)	6-107
3BH Change the Current Directory (CHDIR) ..	6-108
3CH Create a File (CREAT)	6-109
3DH (DOS 2.10) Open a File	6-110
3DH (DOS 3.00 and 3.10) Open a File	6-112
3EH Close a File Handle	6-122
3FH Read from a File or Device	6-123
40H Write to a File or Device	6-125

41H Delete a File from a Specified Directory
(UNLINK) 6-127

42H Move File Read Write Pointer (LSEEK) .. 6-128

43H Change File Mode (CHMOD) 6-130

44H I/O Control for Devices (IOCTL) 6-132

45H Duplicate a File Handle (DUP) 6-139

46H Force a Duplicate of a Handle (FORCDUP) 6-140

47H Get Current Directory 6-141

48H Allocate Memory 6-142

49H Free Allocated Memory 6-143

4AH Modify Allocated Memory Blocks
(SETBLOCK) 6-144

4BH Load or Execute a Program (EXEC) 6-145

4CH Terminate a Process (EXIT) 6-150

4DH Get Return Code of a Subprocess (WAIT) 6-151

4EH Find First Matching File (FIND FIRST) .. 6-152

4FH Find Next Matching File (FIND NEXT) .. 6-154

54H Get Verify Setting 6-155

56H Rename a File 6-156

57H Get/Set a File's Date and Time 6-158

59H (DOS 3.00 and 3.10) Get Extended Error . 6-160

5AH (DOS 3.00 and 3.10) Create Unique File . 6-162

5BH (DOS 3.00 and 3.10) Create New File ... 6-164

5CH (DOS 3.00 and 3.10) Lock/Unlock File Access	6-165
5E00H (DOS 3.10) Get Machine Name	6-168
5E02H (DOS 3.10) Set Printer Setup	6-169
5E03H (DOS 3.10) Get Printer Setup	6-170
5F02H (DOS 3.10) Get Redirection List Entry .	6-171
5F03H (DOS 3.10) Redirect Device	6-173
5F04H (DOS 3.10) Cancel Redirection	6-176
62H (DOS 3.00 and 3.10) Get Program Segment Prefix Address (PSP)	6-178

Introduction

This chapter contains:

- A list of the registers used by DOS.
- A list of the extended ASCII codes.
- A detailed description of all the interrupts and function calls.

Version Specific Information

The following information in this chapter is specific to a version of DOS:

Interrupts:

DOS version 2.10 supports interrupts 20H to 27H.

DOS version 3.00 supports interrupts 20H to 2FH.

DOS version 3.10 supports interrupts 20H to 2FH.

Function Calls:

DOS version 2.10 supports function calls 00H to 57H.

DOS version 3.00 supports function calls 00H to 5CH and 62H, which includes the following new and changed function calls for DOS 3.00:

- 3DH Open File; supports file sharing
- 59H Get Extended Error
- 5AH Create Temporary File
- 5BH Create New File
- 5CH Lock/Unlock File Access
- 62H Get Program Segment Prefix Address

DOS version 3.10 supports function calls 00H to 62H, which includes the following new function calls for DOS 3.10:

- 5E00H Get Machine Name
- 5E02H Set Printer Setup
- 5E03H Get Printer Setup
- 5F02H Get Redirection List Entry
- 5F03H Redirect Device
- 5F04H Cancel Redirection

For DOS 3.00 and 3.10, interrupt 24H (Critical Error Handler Vector), bits 3–5 of AH indicate which responses from an error are valid. Also, DOS 3.00 and 3.10 handles invalid responses differently than DOS 2.10. Refer to “Handling of Invalid Responses” in this chapter for more information.

DOS Registers

DOS uses the following registers, pointers, and flags when it executes interrupts and function calls.

General Registers	
Register	Definition
AX	Accumulator (16-bit)
AH	Accumulator high-order byte (8-bit)
AL	Accumulator low-order byte (8-bit)
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

Pointers	
Register	Definition
SP	Stack pointer (16-bit)
BP	Base pointer (16-bit)
IP	Instruction pointer (16-bit)

Segment Registers	
Register	Definition
CS	Code segment (16-bit)
DS	Data segment (16-bit)
SS	Stack segment (16-bit)
ES	Extra segment (16-bit)

Index Registers	
Register	Definition
DI	Destination index (16-bit)
SI	Stack index (16-bit)

Extended ASCII Codes

For certain keys or key combinations that cannot be represented in standard ASCII code, an extended ASCII code is returned. The extended ASCII code is returned as the second byte of a 2 byte string. Therefore, if the ASCII value returned is zero, examine the second byte to obtain the extended ASCII code.

The following table lists the extended ASCII codes and their meanings.

Extended ASCII Code	Meaning
3	NUL (null character)
15	Shift tab
16-25	Alt- Q, W, E, R, T, Y, U, I, O, P
30-38	Alt - Z, X, C, V, B, M, N
59-68	Function keys F1 through F10
71	Home
72	Cursor up
73	Page up
75	Cursor left
77	Cursor right
79	End
80	Cursor down
81	Page down
82	Insert
83	Delete
84-93	F11-F20 (Shift F1-F10)
94-103	F21-F30 (Ctrl F1-F10)

Interrupts

We recommend that a program wishing to examine or set the contents of any interrupt vector use the DOS function calls (35H and 25H) provided for those purposes, and avoid referencing the interrupt vector locations directly.

DOS reserves interrupt types 20H to 3FH for its use. This means absolute memory locations 80H to FFH are reserved by DOS. The defined interrupts are as follows with all values in hexadecimal.

20H Program Terminate

Issue interrupt 20H to exit from a program. This vector transfers to the logic in DOS 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 “Function Calls” on page 6-32.

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. Do not issue this interrupt directly, the EXEC function call does this for you.

23H Ctrl-Break Exit Address

If the user enters Ctrl-Break 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 aborted or not. If the carry flag is set, the program is aborted, otherwise execution continues (as with a return by IRET). If the user-written Ctrl-Break interrupt uses functions 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 was made. There are no restrictions on what the Ctrl-Break handler is allowed to do, including DOS function calls, as long as the registers are unchanged if IRET is used.

If the program creates a new segment and loads in a second program, which itself changes the Ctrl-Break address, the termination of the second program and return to the first causes the Ctrl-Break address to be restored to the value it had before execution of the second program. It is restored from the second program's Program Segment Prefix. Do not issue this interrupt directly.

24H Critical Error Handler Vector

When a critical error occurs within DOS, 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 (probably the most common occurrence), bit 7=1 if not.

BP:SI contains the address of a Device Header Control Block from which additional information can be retrieved (see below).

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. These are the error codes:

Error Code	Error Name
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 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
FLAGS	

The registers are set such that if an IRET is executed, DOS responds according to (AL) as follows:

- (AL) =0 ignore the error.
 =1 retry the operation.
 =2 terminate the program
 through interrupt 23H.
 =3 fail the system call
 that is in progress.

Note: Be careful when choosing ignore as a response because this causes DOS to believe that an operation has completed successfully when actually it may not have.

To return control from the critical error handler to a user error routine, the following should be true:

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 gives the appropriate prompt, and waits for the user input (Abort, Retry, or Ignore). After the user input, DOS 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, etc.). AH bits 0-2 indicate the affected disk area and whether it was a read or write operation, as follows:

Bit 0=0 if read operation,
 1 if write operation
 Bits 2-1 (affected disk area)
 0 0 DOS area
 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 (DOS 3.00 and 3.10)

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 ABORT (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, the error code is in DI as above.

Notes:

1. Before giving this routine control for disk errors, DOS performs three retries.
2. For disk errors, this exit is taken only for errors occurring during an 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 function calls. If necessary, it may use calls 01H through 12H. Use of any other call destroys the DOS stack and leaves DOS 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, 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 will be in an unstable state until a function call higher than 12H is issued, therefore not recommended.

8. For DOS 3.00 and 3.10, IGNORE requests (AL=0) are converted to FAIL for critical errors that occur on FAT or DIR sectors.
9. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for information on how to obtain additional error information.
10. For DOS 3.10, IGNORE requests (AL=0) are converted to FAIL requests for network 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)
<p>WORD Attributes:</p> <p>Bit 15 = 1 if character device. = 0 if block device</p> <p>If bit 15 is 1:</p> <p>Bit 0 = 1 if current standard input</p> <p>Bit 1 = 1 if current standard output</p> <p>Bit 2 = 1 if current NULL device</p> <p>Bit 3 = 1 if current CLOCK device</p> <p>Bit 14 is the IOCTL bit</p>
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 Absolute Disk Read

This transfers control directly to the device driver. 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. The request is as follows:

- (AL) Drive number (for example,
0=A or 1=B)
- (CX) Number of sectors to read
- (DX) Beginning logical sector number
- (DS:BX) Transfer address

The number of sectors specified is transferred between the given drive and the transfer address. *Logical sector numbers* 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 then 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 disk, due to 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 zero. If the transfer was not successful CF=1 and (AX) indicate the error as follows. (AL) is the DOS 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

26H Absolute Disk Write

This vector is the counterpart of interrupt 25H above. Except that this is a write, the description above applies.

27H Terminate but Stay Resident

This vector is used by programs that are to remain resident when COMMAND.COM regains control.

DOS function call 31H is the preferred method to cause a program to remain resident, because this allows return information to be passed, and allows a program larger than 64K 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), then execute an interrupt 27H. DOS then considers the program as an extension of DOS, so the program is not overlaid when other programs are executed. This concept is very useful for loading programs such as user-written interrupt handlers 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 64K.
4. Memory can be more efficiently used 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 function call 4CH allows the terminating program to pass a completion (or error) code to DOS, which can be interpreted within batch processing (see function call 31H).
6. Terminate but stay resident programs do not close files.

28H–2EH Reserved for DOS

These interrupts are reserved for DOS use.

2FH Multiplex Interrupt

Interrupt 2FH is the multiplex interrupt. A general interface is defined between two processes. It is up to the specific application using interrupt 2FH to define 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 and the multiplex number is checked to see if any other application is using the same multiplex number. There is no predefined method for assigning a multiplex number to a handler. You must just 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=0 through AH=7FH are reserved for DOS. Applications should use multiplex numbers 80H through FFH.

Note: When in the chain for interrupt 2FH, if your code calls DOS or if you execute with interrupts enabled, your code must be reentrant/recursive.

Function Codes

The following table contains the function codes that you can specify in AL to perform a specific function.

Function Codes	Description
0	Get installed state
1	Submit file
2	Cancel file
4	Status
5	End of status

2FH Error Codes

The following table contains the error codes that are returned from interrupt 2FH.

Error Codes	Description
1	Invalid function
2	File not found
3	Path not found
4	Too many open files
5	Access denied
8	Queue full
9	Busy
12	Name too long
15	Invalid drive

AH = 1 is the resident part of PRINT. It has the following functions:

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. On return, AL contains the installed state as follows:

AL=0 Not installed, O.K. to install

AL=1 Not installed, not O.K. 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 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, 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. On return, DX contains the error count. 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 "2FH Error Codes" on page 6-27.

AL=F8–FF Reserved by DOS

Example 2FH Handler

```

MYNUM          EQU      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,OF8H
    JB       DO_FUNC
    IRET                                ; IRET on reserved functions

```

```

DO_FUNC:

```

```

    OR       AL,AL
    JNE      NON_INSTALL          ; Non Get Installed State rec
    MOV      AL,OFFH             ; Say I'm here
    IRET                                ; All done

```

```

NON_INSTALL:

```

```

.
.
.

```

Installing the Handler

```

MOV     AH,MYNUM
XOR     AL,AL
INT     2FH                               ; Ask if already installed
OR      AL,AL
JZ      OK_INSTALL

BAD_INSTALL:                               ; Handler already installed

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

These interrupts are reserved for DOS use.

Function Calls

DOS provides a wide variety of function calls for character device I/O, file management, memory management, date and time functions, execution of other programs, and others. They are grouped as follows (call numbers are in hexadecimal):

Hex Values	Meaning
0	Program terminate
1-C	Traditional character device I/O
D-24	Traditional file management
25-26	Traditional nondevice functions
27-29	Traditional file management
2A-2E	Traditional nondevice functions
2F-38	Extended function group
39-3B	Directory group
3C-46	Extended file management group
47	Directory group
48-4B	Extended memory management group
4C-4F	Extended function group
54-57	Extended function group
59-5C	Extended function group
5E-5F	Network function group
62	Extended function group

Listing of Function Calls

00H	Program terminate
01H	Keyboard input
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	Print 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
19H	Current disk
1AH	Set disk transfer address
1BH	Allocation table information
1CH	Allocation table information for specific device
1DH	Reserved by DOS
1EH	Reserved by DOS
1FH	Reserved by DOS
20H	Reserved by DOS
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 version number
31H	Terminate process and remain resident
32H	Reserved by DOS
33H	Ctrl-Break check
34H	Reserved by DOS
35H	Get vector
36H	Get disk free space
37H	Reserved by DOS
38H	Set or get 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
50H	Reserved by DOS

51H	Reserved by DOS
52H	Reserved by DOS
53H	Reserved by DOS
54H	Get verify setting
55H	Reserved by DOS
56H	Rename a file
57H	Get/set a file's date and time
58H	Used internally by DOS
59H	Get extended error
5AH	Create temporary file
5BH	Create new file
5CH	Lock/unlock file access
5DH	Reserved by DOS
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
61H	Reserved by DOS
62H	Get PSP address

DOS Internal Stack

When DOS takes control, it switches to an internal stack. User registers are preserved unless information is passed back to the requester as indicated in the specific requests. The user stack needs to be sufficient to accommodate the interrupt system. It is recommended that the user stack be 80H in addition to the user needs.

Error Return Information

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.

If you are using DOS version 2.10, check the error code returned. For a list of error codes returned by function calls when you are using DOS 2.10, refer to “DOS 2.10 Error Codes” in this chapter.

If you are using DOS 3.00 or 3.10, use the Get Extended Error function call to return additional information about the error code. For more information, refer to “Get Extended Error” in this chapter.

DOS 2.10 Error Codes

If you are using function calls 38H–57H with DOS version 2.10, to check if an error has occurred, check for the following error codes in the AX register.

Function Call Number	Error Codes	Function Call Number	Error Codes
38H	2	44H	1,3,5,6
39H	3,5	45H	4,6
3AH	3,5,15	46H	4,6
3BH	3	47H	15
3CH	3,4,5	48H	7,8
3DH	2,3,4,5,12	49H	7,9
3EH	6	4AH	7,8,9
3FH	5,6	4BH	1,2,3,5,8,10,11
40H	5,6	4EH	2,3,18
41H	2,3,5	4FH	18
42H	1,6	56H	2,3,5,17
43H	1,2,3,5	57H	1,6

Get Extended Error (DOS 3.00 and 3.10)

The Get Extended Error function call (59H) is intended to provide a common set of error codes and to supply more extensive information about the error to the application. The information returned from function call 59H, in addition to the error code, is the error class, the locus, and the recommended action. The error class provides information about the error type (hardware, internal, system, etc.). The locus provides information about the area involved in the failure (serial device, block device, network, or memory). The recommended action provides a default action for programs that do not understand the specific error code.

Programs written from now on are expected to use the extended error support both from interrupt 24H hard error handlers and after any interrupt 21H function calls.

•

FCB function calls report an error by returning FFH in AL. Handle function calls report an error by setting the carry flag and returning the error code in AX. Interrupt 21H handle function calls for DOS 2.00 and 2.10 continue to return the error codes 1–18. Interrupt 24H handle function calls continue to return error codes 0–12. But the application can obtain any of the error codes listed in the extended error codes table by issuing function call 59H. Handle function calls for DOS 3.00 and 3.10 can return any of the error codes. However, it is recommended that the function call is followed by function call 59H to obtain the error class, the locus, and the recommended action.

In order to create a common error table, error codes 0–12 from interrupt 24H correspond to error codes 19–31 in the extended error codes table. When a FAIL option is specified in the interrupt 24H error handler, issuing function call 59H returns error code 83 (FAIL on interrupt 24H).

The Extended Error Codes are grouped as follows:

0	No error
01–18	Error mappings for DOS 2.00/2.10 INT 21H errors
19–31	Error mappings for DOS 2.00/2.10 INT 24H errors
32–88	Errors for DOS 3.00/3.10

Note: Do not code to specific error codes. If you encounter an extended error code you do not recognize, perform the recommended action. Refer to “Actions” in this chapter for more information.

Extended Error Codes

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. To obtain information about the error, such as the *error class*, *locus*, and recommended *action*, issue the Get Extended Error function call 59H.

Code	Meaning
1	Invalid function number
2	File not found
3	Path not found
4	Too many open files (no handles left)
5	Access denied
6	Invalid handle
7	Memory control blocks destroyed
8	Insufficient memory
9	Invalid memory block address
10	Invalid environment
11	Invalid format
12	Invalid access code
13	Invalid data
14	Reserved
15	Invalid drive was specified
16	Attempt to remove the current directory
17	Not same device
18	No more files
19	Attempt to write on write-protected diskette
20	Unknown unit
21	Drive not ready
22	Unknown command
23	Data error (CRC)
24	Bad request structure length
25	Seek error
26	Unknown media type
27	Sector not found
28	Printer out of paper
29	Write fault
30	Read fault
31	General failure
32	Sharing violation
33	Lock violation

34	Invalid disk change
35	FCB unavailable
36	Sharing buffer overflow
37-49	Reserved
50	Network request not supported
51	Remote computer not listening
52	Duplicate name on network
53	Network name not found
54	Network busy
55	Network device no longer exists
56	Net BIOS command limit exceeded
57	Network adapter hardware error
58	Incorrect response from network
59	Unexpected network error
60	Incompatible remote adapter
61	Print queue full
62	Not enough space for print file
63	Print file was deleted
64	Network name was deleted
65	Access denied
66	Network device type incorrect
67	Network name not found
68	Network name limit exceeded
69	Net BIOS session limit exceeded
70	Temporarily paused
71	Network request not accepted
72	Print or disk redirection is paused
73-79	Reserved
80	File exists
81	Reserved
82	Cannot make directory entry
83	Fail on INT 24
84	Too many redirections
85	Duplicate redirection
86	Invalid password
87	Invalid parameter
88	Network device fault

Error Classes

This value provides information about the type of error.

Value	Description
1	Out of Resource: Out of space, channels, etc.
2	Temporary Situation: Something that is expected to “go away” with time. Note that this is not an error condition, but a “situation” such as file locked, etc.
3	Authorization: Permission problem.
4	Internal: Internal error in system software. A situation judged to be a system software bug rather than a user or system failure.
5	Hardware Failure: A serious problem not the fault of user program.
6	System Failure: Serious failure of system software. Not directly the fault of the user. For example, configuration files missing or wrong.
7	Application Program Error: Inconsistent requests, etc.
8	Not Found: File/item not found.
9	Bad Format: File/item of invalid format, type, or otherwise invalid or unsuitable.
10	Locked: File/item interlocked.
11	Media: Media failure (wrong disk, CRC error...). Wrong disk in drive, bad spot on media, etc.
12	Already Exists: Collision with existing item, such as trying to declare a machine name that already exists.
13	Unknown: Classification doesn't exist or is inappropriate.

Actions

Note that these are recommended actions. In the most critical cases, the application will analyze the error codes and take specific action. These defaults are for programs that do not understand the specific error code.

Value	Description
1	Retry: Retry a few times, then prompt user to determine if the program should continue or be aborted.
2	Delay Retry: Retry after pause (a few times), then prompt user to determine if the program should continue or be aborted.
3	User: Ask user to reenter input. Typically, a bad drive letter or bad filename was presented in the system call. Naturally, if the value was "built into" the program and not directly keyed in by the user, then the program would not, in fact, "ask the user to reenter input." This action means that if the data came from a user, the best action is to tell him to try again.
4	Abort: Abort application with cleanup. The application cannot proceed, but the system is sufficiently healthy that the application should try an orderly shutdown.
5	Immediate Exit: Abort application immediately, skip cleanup. We do not recommend that the application try to close files, update indexes, but that it exit as soon as possible.
6	Ignore: Ignore.
7	Retry After User Intervention: The user needs to perform some action (like taking out a diskette and putting in a different one); then the operation should be retried.

Locus

This value provides additional information to help locate the area involved in the failure.

Value	Description
1	Unknown: Nonspecific. Not appropriate.
2	Block Device: Related to random access mass storage (disk).
3	Net: Related to the network.
4	Serial Device: Related to serial devices.
5	Memory: Related to random access memory.

ASCIIZ Strings

Several of the function calls accept an ASCIIZ string as input. This consists of an ASCII string containing an optional drive specifier, followed by a directory path and in some cases a filename. The string is terminated by a byte of binary zeros. For example:

```
B:\LEVEL1\LEVEL2\FILE1
```

followed by a byte of zeros.

The maximum size of an ASCIIZ string is 128 bytes, including the drive, colon, and null terminator.

Note: All function calls that accept path names accept a forward slash or a backslash as a path separator character.

Network Paths

For DOS 3.10, several of the function calls accept a network path as input if the IBM PC Network is loaded. A network path consists of an ASCII string containing a computer name, followed by a directory path, and in some cases a filename. The string cannot contain a drive specifier. The string is terminated by a byte of binary zeros. For example,

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

All function calls that accept an ASCII path as input, also accept a network path as input. Two function calls that do not accept a network path as input are Change Current Directory (3BH) and Find First Matching File (4EH).

Network Access Rights

The explanation of some function calls contains a section under remarks called "Network Access Rights." Any information under "Network Access Rights" tells you the access requirements for a directory that a computer on the network needs to be able to execute the function call when using DOS 3.10. For example, suppose you want to execute function call 5BH (Create New File). 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 (no Create access), you cannot create a file in the directory.

File Handles

The extended function calls (3CH-62H) that supporting files or devices use an identifier known as a “handle.” When you create or open a file or device with these calls, a 16-bit binary value is returned in AX. This is the handle (sometimes known as a token) that you will use in referring to the file after it’s been opened.

The following handles are predefined by DOS and can be used by your program. You do not need to open them before using them:

Hex Value	Meaning
0000	Standard input device. Input can be redirected.
0001	Standard output device. Output can be redirected.
0002	Standard error output device. Output cannot be redirected.
0003	Standard auxiliary device.
0004	Standard printer device.

Using DOS Functions

Most of the function calls require input to be passed to them in registers. After setting the proper register values, the function may be used in one of these ways:

1. Place the function number in AH and execute a long call to offset 50H in your program segment prefix.
2. Place the function number in AH and issue interrupt type 21H. This is the preferred method of using DOS function calls.
3. There is an additional mechanism provided for preexisting programs that were written with different calling conventions. This method should be avoided for all new programs. The function number is placed in the CL register and other registers are set according to the function specification. Then an intrasegment call is made to location 5 in the current code segment. That location contains a long call to the DOS function dispatcher. Register AX is always destroyed if this mechanism is used; otherwise, it is the same as normal function calls. This method is valid only for function calls (00H-24H).

Notes:

1. All FCB function calls do not allow invalid characters (0DH-29H).
2. Device names cannot end in a colon (:).
3. The contents of the AX register may be altered by any of the function calls. Even if no error code is returned in AX, the user cannot be guaranteed that AX is unchanged.
4. Function calls 01H through 0CH use the standard devices listed in the "File Handles" section. Refer to "File Handles" on page 6-46 for more information.

00H

Program Terminate

Purpose: Terminates the execution of a program.

On Entry	Register Contents
AH	00H
CS	Points to PSP

On Return	Register Contents
	NONE

Remarks: 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.

01H

Keyboard Input

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.

On Entry	Register Contents
AH	01H

On Return	Register Contents
AL	Character from the standard input device

Remarks: The character is checked for a Ctrl-Break. If Ctrl-Break is detected, an interrupt 23H is executed.

Note: 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. Refer to "Extended ASCII Codes" in the beginning of this chapter for a table of Extended ASCII codes.

02H

Display Output

Purpose: Outputs the character in DL to the standard output device.

On Entry	Register Contents
AH	02H
DL	Character

On Return	Register Contents
	NONE

Remarks: If the character in DL is a backspace (08), the cursor is moved left on position (nondestructive). If a Ctrl-Break is detected after the output, an interrupt 23H is executed.

03H

Auxiliary Input

Purpose: Waits for a character from the standard auxiliary device, then returns that character in AL.

On Entry	Register Contents
AH	03H

On Return	Register Contents
AL	Character from the auxiliary device

Remarks: Auxiliary (AUX, COM1, COM2) support is unbuffered and noninterrupt driven.

At startup, DOS 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, it is recommended that you use the ROM BIOS routine (interrupt 14H) or write an AUX device drivers and use IOCTL.

04H

Auxiliary Output

Purpose: Outputs the character in DL to the standard auxiliary device.

On Entry	Register Contents
AH	04H
DL	Character

On Return	Register Contents
	NONE

05H

Printer Output

Purpose: Outputs the character in DL to the standard printer device.

On Entry	Register Contents
AH	05H
DL	Character

On Return	Register Contents
	NONE

06H

Direct Console I/O

Purpose: Waits for a character from the standard input device if one is ready.

On Entry	Register Contents
AH	06H
DL	FFH, for console input 00H-FEH, for console output

On Return	Register Contents
AL	See description below

Remarks: If DL is FFH, AL returns with the zero flag clear and an input character from the standard input device if one is ready. If a character is not ready, the zero 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.

Note: 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. Refer to “Extended ASCII Codes” in the beginning of this chapter for a table of Extended ASCII codes.

07H

Direct Console Input Without Echo

Purpose: Waits for a character to be read at the standard input device (unless one is ready), then returns the character in AL.

On Entry	Register Contents
AH	07H

On Return	Register Contents
AL	Character from standard input device

Remarks: As with function call 06H, no checks are made on the character.

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.

On Entry	Register Contents
AH	08H

On Return	Register Contents
AL	Character from standard input device

Remarks: The character is checked for Ctrl-Break. If Ctrl-Break is detected, an interrupt 23H is executed.

Note: 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. Refer to "Extended ASCII Codes" in the beginning of this chapter for a table of Extended ASCII codes.

09H

Print String

Purpose: Outputs the characters in the print string to the standard output device.

On Entry	Register Contents
AH	09H
DS:DX	Pointer to the character string

On Return	Register Contents
	NONE

Remarks: 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.

0AH

Buffered Keyboard Input

Purpose: Reads characters from the standard input device and places them in the buffer beginning at the third byte.

On Entry	Register Contents
AH	0AH
DS:DX	Pointer to an input buffer

On Return	Register Contents
	NONE

Remarks: The first byte of the input buffer specifies the number of characters the buffer can hold. This value cannot be zero. 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.

0BH

Check Standard Input Status

Purpose: Checks if there is a character available from the standard input device.

On Entry	Register Contents
AH	0BH

On Return	Register Contents
AL	FFH If the character is available from the standard input device 00H If no character is available from the standard input device

Remarks: If a character is available from the standard input device, AL is FFH. Otherwise, AL is 00H. 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 buffer of any pretyped characters, then executes the function call number in AL (only 01H, 06H, 07H, 08H, and 0AH are allowed).

On Entry	Register Contents
AH	0CH
AL	Function number

On Return	Register Contents
	NONE

Remarks: This forces the system to wait until a character is typed.

0DH

Disk Reset

Purpose: Flushes all file buffers.

On Entry	Register Contents
AH	0DH

On Return	Register Contents
	NONE

Remarks: Files changed in size but not closed are not properly recorded in the disk directory.

0EH

Select Disk

Purpose: Selects the drive specified in DL (0=A, 1=B, etc.) (if valid) as the default drive.

On Entry	Register Contents
AH	0EH
DL	Drive number (0=A, 1=B, etc.)

On Return	Register Contents
AL	Total number of drives

Remarks: The number of drives (total of diskette and fixed disk drives) is returned in AL. For DOS 3.00 and 3.10, the minimum value returned in AL is 5. 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.

0FH

Open File

Purpose: Searches the current directory for the named file and AL returns FFH if it is not found. If it is found, AL returns 00H and the FCB is filled as described below.

On Entry	Register Contents
AH	0FH
DS:DX	Pointer to an unopened FCB

On Return	Register Contents
AL	00H If file opened FFH If file not opened

Remarks: If the drive code was 0 (default drive), it is changed to the actual drive used (1=A, 2=B, etc.). This allows changing the default drive without interfering with subsequent operations on this file. The current block field (FCB bytes C-D) is set to zero. The size of the record to be worked with (FCB bytes E-F) is set to the system default of 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 in this chapter.

10H

Close File

Purpose: Closes a file after a file write.

On Entry	Register Contents
AH	10H
DS:DX	Pointer to an opened FCB

On Return	Register Contents
AL	00H If the file is found FFH If the file is not found in the current directory

Remarks: This function call must be done on open files that are no longer needed, and after file writes to ensure all directory information is updated. If the file is not found in its correct position in the current directory, it is assumed the diskette 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 for the first matching filename.

On Entry	Register Contents
AH	11H
DS:DX	Pointer to an unopened FCB

On Return	Register Contents
AL	00H If matching filename found FFH If matching filename was not found

Remarks: The current disk directory is searched for the first matching filename. If none are found, AL returns FFH. For DOS 2.10, ?s are allowed in the filename. For DOS 3.00 and 3.10, global filename characters are allowed. 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, then the first byte at the disk transfer address is set to FFH followed by 5 bytes of zeros, 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.

11H

Search for First Entry

- 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 FCB attribute byte is zero, 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.

The attribute bits are defined in "DOS Disk Directory" on page 5-10.

12H

Search for Next Entry

Purpose: Searches the current directory for the next matching filename.

On Entry	Register Contents
AH	12H
DS:DX	Pointer to an the unopened FCB specified from the previous Search First (11H) or Search Next (12H).

On Return	Register Contents
AL	00H If matching filename found FFH If matching filename not found

Remarks: 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. For DOS 2.10, ?s are allowed in the filename. For DOS 3.00 and 3.10, global filename characters are allowed.

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 with this FCB 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.

On Entry	Register Contents
AH	13H
DS:DX	Pointer to an unopened FCB

On Return	Register Contents
AL	00H File deleted FFH If directory entry match was not found

Remarks: All matching current directory entries are deleted. The global filename character “?” is allowed in the filename. 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.

14H

Sequential Read

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 incremented.

On Entry	Register Contents
AH	14H
DS:DX	Pointer to an opened FCB

On Return	Register Contents
AL	00H If read was successfully completed 01H If EOF (no data read) 02H If DTA too small (read canceled) 03H If EOF (a partial record was read and filled out with zeros)

Remarks: 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 incremented.

On Entry	Register Contents
AH	15H
DS:DX	Pointer to an opened FCB

On Return	Register Contents
AL	00H If write was successfully completed 01H If diskette is full (write canceled) 02H If DTA too small (write canceled)

Remarks: If the file is specified in read-only mode, the sequential write is not performed.

Network Access Rights: Requires Write access rights.

16H

Create File

Purpose: Searches the current directory of the specified drive for a matching entry.

On Entry	Register Contents
AH	16H
DS:DX	Pointer to an unopened FCB

On Return	Register Contents
AL	00H If file created (matching entry found or empty entry found)
	FFH If file not created (full directory or disk and no matching directory entry)

Remarks: 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 zero-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.)

On Entry	Register Contents
AH	17H
DS:DX	Pointer to a modified FCB

On Return	Register Contents
AL	00H If file renamed (matching filename found) FFH If no matching filename found or if an attempt to rename an existing filename

Remarks: The modified FCB has a drive code and filename in the usual position, and a second filename starting 6 bytes after the first (DS:DX+11H) in what is normally a reserved area. If “?”s appear in the second name, then 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.

On Entry	Register Contents
AH	19H

On Return	Register Contents
AL	Current default drive (0=A, 1=B, etc.)

Remarks: AL returns with the code of the current default drive (0=A, 1=B, etc.).

1AH

Set Disk Transfer Address

Purpose: Sets the disk transfer address to DS:DX.

On Entry	Register Contents
AH	1AH
DS:DX	Disk transfer address

On Return	Register Contents
	NONE

Remarks: DOS 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.

Note: You can get the DTA using function call 2FH.

1BH

Allocation Table Information

Purpose: Returns information about the allocation table for the default drive.

On Entry	Register Contents
AH	1BH

On Return	Register Contents
DS:BX	Pointer to the media descriptor byte for the default drive
DX	Number of allocation units
AL	Number of sectors/allocation unit
CX	Size of the physical sector

Remarks: For more information on DOS disk allocation, refer to "DOS Disk Directory" on page 5-10. Also, 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.

On Entry	Register Contents
AH	1CH
DL	Drive number

On Return	Register Contents
DS:BX	Points to the media descriptor byte of the drive specified in DL
AL	Number of sectors/allocation unit
DX	Number of allocation units
CX	Size of the physical sector

Remarks: This call is identical to call 1BH except that, on entry, DL contains the number of the drive that contains the needed information (0 = default, 1 = A, etc.). For more information on DOS disk allocation, refer to "DOS Disk Directory" on page 5-10. 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.

On Entry	Register Contents
AH	21H
DS:DX	Pointer to an opened FCB

On Return	Register Contents
AL	00H If read was successfully completed 01H If EOF (no data read) 02H If DTA too small (read canceled) 03H If EOF (a partial record was read and filled out with zeros)

Remarks: The current block and current record fields are set to agree with the random record field. Then the record addressed by these fields is read into memory at the current disk transfer address.

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.

On Entry	Register Contents
AH	22H
DS:DX	Pointer to an opened FCB

On Return	Register Contents
AL	00H If write was successfully completed 01H If diskette is full (write canceled) 02H If DTA too small (write canceled)

Remarks: 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 diskette directory for an entry that matches the specified file and sets the FCBs random record field to the number of records in the file.

On Entry	Register Contents
AH	23H
DS:DX	Pointer to an unopened FCB

On Return	Register Contents
AL	00H If the directory entry is found FFH If the directory entry not found

Remarks: The diskette 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.

24H**Set Relative Record Field**

Purpose: Sets the random record field to the same file address as the current block and record fields.

On Entry	Register Contents
AH	24H
DS:DX	Pointer to an opened FCB

On Return	Register Contents
	NONE

Remarks: You must call this function before you perform random read and writes, and random block read and writes.

25H

Set Interrupt Vector

Purpose: Sets the interrupt vector table for the interrupt number.

On Entry	Register Contents
AH	25H
DS:DX	Address of interrupt handling routine
AL	Interrupt number

On Return	Register Contents
	NONE

Remarks: The interrupt vector table for the interrupt number specified in AL is set to address contained in DS:DX. Use function call 35H (Get Vector) to obtain the contents of the interrupt vector.

26H

Create New Program Segment

Purpose: Creates a new program segment.

On Entry	Register Contents
AH	26H
DX	Segment number for the new program segment

On Return	Register Contents
	NONE

Remarks: 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 terminates.

Note: You should avoid using this call. We recommend that you use the EXEC function call 4PH instead.

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.

On Entry	Register Contents
AH	27H
DS:DX	Pointer to an opened FCB
CX	Number of records to be read

On Return	Register Contents
AL	00H If read was successfully completed
	01H If EOF (no data read)
	02H If DTA too small (read canceled)
	03H If EOF (a partial record was read and filled out with zeros)
CX	Actual number of records read

Remarks: The random record field and the current block/record fields are set to address the next record (the first record not read).

Network Access Rights: Requires Read access rights.

28H

Random Block Write

Purpose: Writes the specified number of records from the file address specified by the random record field into the disk transfer address.

On Entry	Register Contents
AH	28H
DS:DX	Pointer to an opened FCB
CX	Number of records to be written

On Return	Register Contents
AL	00H If write was successfully completed
	01H If diskette is full (write canceled)
	02H If DTA too small (write canceled)
CX	Actual number of records written

Remarks: If there is insufficient space on the disk, AL returns 01H and no records are written. If CX is zero upon entry, no records are written, but the file is set to the length specified by the random record field, whether longer or shorter than the current file size. (Allocation units are released or allocated as appropriate.)

Network Access Rights: Requires Write access rights.

29H

Parse Filename

Purpose: Parses the specified filename.

On Entry	Register Contents
AH	29H
DS:SI	Pointer to a command line to parse
ES:DI	Pointer to a portion of memory that will be filled with an unopened FCB
AL	Bit value controls parsing

On Return	Register Contents
AL	00H If no global filename characters in command line 01H If global filename characters used in command line FFH If drive specifier invalid
DS:SI	Points to the first character after the parsed filename
ES:DI	Points to the first byte of the formatted FCB

Remarks: 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

29H

Parse Filename

If bit 0 = 1, then leading separators are scanned off the command line at DS:SI. Otherwise, no scan-off of leading separators takes place.

If bit 1 = 1, then the drive ID byte in the result FCB will be set (changed) *only* if a drive was specified in the command line being parsed.

If bit 2 = 1, then the filename in the FCB will be changed only if the command line contains a filename.

If bit 3 = 1, then the filename extension in the FCB will be changed only if the command line contains a filename extension.

Filename separators include the following characters : . ; , = + plus TAB and SPACE. Filename terminators include all of these characters plus < , > , ! , / , " , [,] , and any control characters.

The command line is parsed for a filename of the form *d:filename.ext*, and if found, a corresponding unopened FCB is created at ES:DI. If no drive specifier is present, it is assumed to be all blanks. If the character * appears in the filename or extension, then it and all remaining characters in the name or extension are set to ?.

If either ? or * appear in the filename or extension, AL returns 01H; if the drive specifier in AL returns FFH; otherwise 00H.

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, year, month and date.

On Entry	Register Contents
AH	2AH

On Return	Register Contents
AL	Day of the week (0=SUN 6=SAT)
CX	Year (1980 - 2099)
DH	Month (1 - 12)
DL	Day (1 - 31)

Remarks: 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. Unless you are using the IBM ROM which ignores date rollovers past the first.

2BH

Set Date

Purpose: Sets the date.

On Entry	Register Contents
AH	2BH
CX	Year (1980 - 2099)
DH	Month (1 - 12)
DL	Day (1 - 31)

On Return	Register Contents
AL	00H, if the date was valid FFH, if the date not valid

Remarks: 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.

On Entry	Register Contents
AH	2CH

On Return	Register Contents
CH	Hour (0 -23)
CL	Minutes (0 - 59)
DH	Seconds (0 - 59)
DL	Hundredths (0 - 99)

Remarks: 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 has the hours (0-23), CL has minutes (0-59), DH has seconds (0-59), DL has 1/100 seconds (0-99). This format is readily converted to a printable form yet can also be used for calculations, such as subtracting one time value from another.

2DH

Set Time

Purpose: Sets the time.

On Entry	Register Contents
AH	2DH
CH	Hour (0 -23)
DH	Seconds (0 - 59)
CL	Minutes (0 - 59)
DL	Hundredths (0 - 99)

On Return	Register Contents
AL	00H, if the time was valid FFH, if the time not valid

Remarks: 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 aborted and AL returns FFH. If the time is valid, AL returns 00H.

2EH

Set/Reset Verify Switch

Purpose: Sets the verify switch.

On Entry	Register Contents
AH	2EH
AL	00H, to set verify off 01H, to set verify on

On Return	Register Contents
	NONE

Remarks: On entry, AL must contain 01H to turn verify on, or 00H to turn verify off. When verify is on, DOS 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.

On Entry	Register Contents
AH	2FH

On Return	Register Contents
ES:BX	The current DTA

Remarks: 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.

On Entry	Register Contents
AH	30H

On Return	Register Contents
BX	0000H
CX	0000H
AL	Major version number
AH	Minor version number

Remarks: 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. For example, for DOS 3.10, the major version number is 03H and the minor version number is 0AH.

Note: If AL returns a major version number of zero, then it can be assumed 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.

On Entry	Register Contents
AH	31H
AL	Return code
DX	Memory size in paragraphs

On Return	Register Contents
	NONE

Remarks: 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.

Note: Memory can be more efficiently used 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).

33H

Ctrl-Break Check

Purpose: Set or get the state of BREAK (Ctrl-Break checking).

On Entry	Register Contents
AH	33H
AL	00H, to request current state 01H, to set the current state
DL	00H, to set current state OFF 01H, to set current state ON

On Return	Register Contents
DL	The current state (00H=OFF, 01H=ON)

Remarks: On entry, AL contains 00H to request the current state of Ctrl-Break checking, 01H to set the state. If setting the state, DL must contain 00H for OFF or 01H for ON. On return, if requesting the current state, DL contains the current state (00H = OFF, 01H = ON).

35H

Get Vector

Purpose: Points to the interrupt handling routine.

On Entry	Register Contents
AH	35H
AL	Interrupt number

On Return	Register Contents
ES:BX	Pointer to the interrupt handling routine.

Remarks: 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.

36H

Get Disk Free Space

Purpose: Returns the disk free space (available clusters, clusters/drive, bytes/sector).

On Entry	Register Contents
AH	36H
DL	Drive (0=default, 1=A)

On Return	Register Contents
BX	Available clusters
DX	Clusters/drive
CX	Bytes/sector
AX	FFFFH if the drive in DL is invalid, otherwise the number of sectors per cluster

Remarks: If the drive number in DL was valid, BX contains the number of available allocation units (clusters), DX contains the total number of clusters on the drive, CX contains the number of bytes per sector, and AX contains the number of sectors per cluster.

Note: This call returns the same information in the same registers (except for the FAT pointer) as the get FAT pointer call (1BH).

38H (DOS 2.10)

Return Country Dependent Information

Purpose: Returns country dependent information.

On Entry	Register Contents
AH	38H
DS:DX	Pointer to the 32-byte memory area
AL	Equals the function code

On Return	Register Contents
AX	Error code if carry flag set
DS:DX	Country data if carry flag not set

Remarks: On entry, DS:DX points to a 32-byte block of memory in which returned information is passed and AL contains a function code. In DOS 2.10, this function code must be 0. The following information is pertinent to international applications:

WORD date/time format
BYTE ASCIIZ string currency symbol followed by byte of zeros
BYTE ASCIIZ string thousands separator followed by byte of zeros
BYTE ASCIIZ string decimal separator followed by byte of zeros
24 bytes Reserved

38H (DOS 2.10)

Return Country Dependent Information

The time and date format has the following values and meaning:

0 = USA standard h:m:s m/d/y

1 = Europe standard h:m:s d/m/y

2 = Japan standard h:m:s d:m:y

38H (DOS 3.00 and 3.10)

Get or Set Country Dependent Information

Purpose: Returns country dependent information.

Get Current Country

On Entry	Register Contents
AH	38H
DS:DX	Pointer to the memory buffer where the data will be returned
AL	00H ; to get current country information Country code ; to get information for countries with a code <255 FFH ; to get country information for countries with a code ≥255
BX	16 bit country code ; if AL=FFH

On Return	Register Contents
AX	Error code if carry flag set
DS:DX	Filled with the country information (described below)
BX	Country code

38H (DOS 3.00 and 3.10)

Get or Set Country Dependent Information

Set Current Country

On Entry	Register Contents
AH	38H
DX	FFFFH
AL	Country code for countries with a code <255 FFH for countries with a code ≥ 255
BX	16-bit country code ; if AL=FFH

On Return	Register Contents
AX	Error code if carry flag set

38H (DOS 3.00 and 3.10)

Get or Set Country Dependent Information

Country Information

WORD Date format
5 BYTE currency symbol null terminated
2 BYTE thousands separator null terminated
2 BYTE decimal separator null terminated
2 BYTE date separator null terminated
2 BYTE time separator null terminated
1 BYTE bit field currency format Bit 0 = 0 if the currency symbol precedes the value = 1 if the currency symbol is after the value Bit 1 = number of spaces between the value and the currency symbol (0 or 1)
1 BYTE number of significant decimal digits in currency
1 BYTE time format Bit 0 = 0 if 12-hour clock Bit 0 = 1 if 24-hour clock
2 WORDS Case map call address
2 BYTES Data list separator null terminated
5 WORDS Reserved

38H (DOS 3.00 and 3.10)

Get or Set Country Dependent Information

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.

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

38H (DOS 3.00 and 3.10)

Get or Set Country Dependent Information

Remarks: Error codes are returned in AX. Issue function call 59H “Get Extended Error” for additional information about the error class, suggested action, and locus. Refer to “Error Return Information” on page 6-36 and “Extended Error Codes” on page 6-40 for more information on the codes returned from function call 59H.

When an alternate keyboard handler is invoked, the keyboard routine is loaded into user memory starting at the lowest portion of available user memory. The BIOS interrupt vector that services the keyboard is changed by the routine to redirect the CPU to the section of user memory where the new keyboard routine now resides. Each keyboard routine takes up approximately 1.6K bytes of read/write memory, and has lookup tables that return ASCII values unique to each language. Refer to the KEYBxx command in the *DOS Reference*.

Once the keyboard interrupt vector is changed by the DOS keyboard routine, the interrupt is always serviced by the routine in read/write memory. Return to the U.S. English keyboard format is available by holding the Ctrl and Alt keys and pressing F1 at the same time. This does not change the interrupt vector back to the BIOS location. In this case, the interrupt is still processed by the read/write routine, but the lookup to convert scan codes to ASCII codes is done in the ROM locations. However, Ctrl–Alt–F1 does not return you to a U.S. keyboard if you are using a computer with ROM keyboard support. Similarly, holding the Ctrl and Alt keys and pressing F2 causes a return to the read/write lookup tables.

39H

Create Subdirectory (MKDIR)

Purpose: Creates the specified directory.

On Entry	Register Contents
AH	39H
DS:DX	Pointer to an ASCIIZ string

On Return	Register Contents
AX	Error codes if carry flag is set

Remarks: On entry, DS:DX contains the address of an ASCIIZ string with drive and directory path names. If any member of the directory path does not exist, then the directory path is not created. On return, a new directory is created at the end of the specified path.

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

Network Access Rights: Requires Create access rights.

3AH**Remove Subdirectory (RMDIR)**

Purpose: Removes the specified directory.

On Entry	Register Contents
AH	3AH
DS:DX	Pointer to an ASCIIZ string

On Return	Register Contents
AX	Error codes if carry flag is set

Remarks: 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 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

Network Access Rights: Requires Create access rights.

3BH

Change the Current Directory (CHDIR)

Purpose: Changes the current directory to the specified directory.

On Entry	Register Contents
AH	3BH
DS:DX	Pointer to an ASCIIZ string

On Return	Register Contents
AX	Error codes if carry flag is set

Remarks: 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, then 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

3CH**Create a File (CREAT)**

Purpose: Creates a new file or truncates an old file to zero length in preparation for writing.

On Entry	Register Contents
AH	3CH
DS:DX	Pointer to an ASCIIZ string
CX	Attribute of the file

On Return	Register Contents
AX	Error codes if carry flag is set 16-bit handle if carry flag not set

Remarks: If the file did not exist, then 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, and the handle is returned in AX. Note that the change mode function call (43H) can later be used 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

Network Access Rights: Requires Create access rights.

3DH (DOS 2.10)

Open a File

Purpose: Opens the specified file.

On Entry	Register Contents
AH	3DH
DS:DX	Pointer to an ASCIIZ path name
AL	Access Code

On Return	Register Contents
AX	Error codes if carry flag is set 16-bit file handle if carry flag not set

Remarks: This call opens any normal or hidden file whose name matches the name specified. Files that end with a colon are not opened.

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 through 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.

3DH (DOS 2.10)

Open a File

Access Codes

AL = 0 File is opened for reading

AL = 1 File is opened for writing

AL = 2 File is opened for both reading and writing

3DH (DOS 3.00 and 3.10)

Open a File

Purpose: Opens the specified file.

On Entry	Register Contents
AH	3DH
DS:DX	Pointer to an ASCIIZ path name
AL	Open mode

On Return	Register Contents
AX	Error codes if carry flag is set 16-bit file handle if carry flag not set

Remarks: 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 through 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

3DH (DOS 3.00 and 3.10)

Open a File

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

Notes:

1. This call opens any normal or hidden file whose name matches the name specified. Files that end with a colon are not opened.
2. When a file is closed, any sharing restrictions placed on it by the open are canceled.
3. File sharing must be loaded for the sharing modes to function. Refer to the SHARE command in Chapter 7 “DOS Commands” of the *DOS Reference*.
4. The file read-only attribute can be set when creating the file using extended FCBs or specifying the appropriate attribute in CX for the handle created by using the CHMOD interrupt 21 function call or the DOS ATTRIB command.
5. If the file is inherited by the child process, all sharing and access restrictions are also inherited.
6. If an open file handle is duplicated by either of the DUP function calls, all sharing and access restrictions are also duplicated.

3DH (DOS 3.00 and 3.10)

Open a File

Open Mode

The open mode is defined in AL and consists of four bit-oriented fields. They are the:

- Inheritance flag
- Sharing mode field
- Reserved field
- Access field

The inheritance flag specifies if the opened file will be inherited by a child process. The access field defines what operations this process may perform on the file. The sharing mode field defines what operations other processes may 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

3DH (DOS 3.00 and 3.10)

Open a File

I Inheritance flag
If I = 0; File is inherited by child processes
If I = 1; File is private to the current process

S Sharing Mode

The file is opened as follows:

If S = 000; Compatibility mode

If S = 001; Deny Read/Write mode (Exclusive)

If S = 010; Deny Write mode

If S = 011; Deny Read mode

If S = 100; Deny None mode

Any other combinations are invalid.

When opening a file, it is important to inform DOS what operations other processes may perform on this file (sharing mode). The default (compatibility mode) denies all other processes access to the file. Perhaps it is all right for other processes to continue to read this file while your process is operating on the file. In this case, you should specify Deny/Write, which inhibits writing by other processes, but allows reading by them.

3DH (DOS 3.00 and 3.10)

Open a File

Similarly, it is important to specify what operations you process will perform (access mode). The default access mode (Read/Write) causes the open request to fail if another process has the file opened with any sharing mode other than deny none. If however, all you intended to do is read from the file, your open will succeed unless all other processes have specified deny none or deny write (therefore increasing access to the file). File sharing requires cooperation of both sharing processes. This cooperation is communicated through the sharing and access mode.

R Reserved (set third 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.

3DH (DOS 3.00 and 3.10)

Open a File

Sharing Modes

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 currently open in Deny Write 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 will indicate Drive not ready, and the extended error will indicate that there was a Sharing violation.

The sharing modes for a file opened in compatibility mode are changed by DOS depending on the read-only attribute of the file. This is to allow sharing of read-only files.

3DH (DOS 3.00 and 3.10)

Open a File

Read-Only		
File Opened By	Access	Sharing Mode
FCB	Read Only	Deny Write
Handle Read	Read Only	Deny Write
Handle Write	Error	-----
Handle Read/Write	Error	-----

Not Read-Only		
File Opened By	Access	Sharing Mode
FCB	Read/Write	Compatibility
Handle Read	Read	Compatibility
Handle Write	Write	Compatibility
Handle Read/Write	Read/Write	Compatibility

3DH (DOS 3.00 and 3.10)

Open a File

Deny Read/Write Mode (Exclusive)

If a file is successfully opened in Deny Read/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.

Deny Write Mode

A file successfully opened in Deny Write 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 Deny Write mode is unsuccessful if the file is currently open with a write access.

Deny Read Mode

A file successfully opened in Deny Read 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 Deny Read sharing mode is unsuccessful if the file is currently open in Compatibility mode or with a read access.

3DH (DOS 3.00 and 3.10)

Open a File

Deny None Mode

A file successfully opened in Deny None mode, places no restrictions on the read/write accessibility of the file. An attempt to open a file in Deny None mode is unsuccessful if the file is currently open in Compatibility mode.

Note: 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:

- Deny Read
- Deny None
- Open for Read/Write access and Deny Write
- Open for Write only and Deny Write

Therefore, in a network environment, Deny Read/Write sharing mode, Compatibility sharing mode, and Input Deny Write 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 (DOS 3.00 and 3.10)

Open a File

2ND, 3RD, ... OPEN

		DRW			DW			DR			ALL			
		I	IO	O	I	IO	O	I	IO	O	I	IO	O	
1 S T O P E N	D R W	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
	D W	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
	D R	I	N	N	N	N	N	Y	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
	A L L	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 :Deny Read/Write Mode (Exclusive)
 DW :Deny Write Mode
 DR :Deny Read Mode
 RW :Read/Write Mode
 I :Read Only Access
 O :Write Only Access
 I/O :Read/Write Access

3EH

Close a File Handle

Purpose: Closes the specified file handle.

On Entry	Register Contents
AH	3EH
BX	File handle returned by open or create

On Return	Register Contents
AX	Error codes if carry flag set NONE if carry flag not set

Remarks: 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 locus. Refer to “Error Return Information” on page 6-36 and “Extended Error Codes” on page 6-40 for more information on the codes returned from function call 59H.

3FH**Read from a File or Device**

Purpose: Transfers the specified number of bytes from a file into a buffer location.

On Entry	Register Contents
AH	3FH
BX	File handle
DS:DX	Buffer address
CX	Number of bytes to be read

On Return	Register Contents
AX	Number of bytes read Error codes if carry flag set

Remarks: 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, reading from the keyboard reads at most one line of text. If this read is performed from the standard input device, the input can be redirected. See "Redirection of Standard Input and Output" in the *DOS Reference*. If the value in AX is 0, then the program has tried to read from the end of file.

3FH

Read from a File or Device

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

Network Access Rights: Requires Read access rights.

40H

Write to a File or Device

Purpose: Transfers the specified number of bytes from a buffer into a specified file.

On Entry	Register Contents
AH	40H
BX	File handle
DS:DX	Address of the data to write
CX	Number of bytes to write

On Return	Register Contents
AX	Number of bytes written Error codes if carry flag set

Remarks: 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 this value is not the same as the number requested, it should be considered an error (no error code is returned, but your program can compare these values). The usual reason for this is a full disk. If this write is performed to the standard output device, the output can be redirected. See "Redirection of Standard Input and Output" in the *DOS Reference*.

40H

Write to a File or Device

To truncate a file at the current position of the file pointer, set the number of bytes (CX) to zero 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).

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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

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.

On Entry	Register Contents
AH	41H
DS:DX	Address of an ASCIIZ string

On Return	Register Contents
AX	Error codes if carry flag set NONE if carry flag not set

Remarks: 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

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.

On Entry	Register Contents
AH	42H
CX:DX	Distance (offset) to move in bytes
AL	Method of moving (0, 1, 2)
BX	File handle

On Return	Register Contents
AX	Error codes if carry flag set
DX:AX	New pointer location if carry flag not set

Remarks: On entry, AL contains a method value. BX contains the file handle. CX:DX contains the desired offset in bytes (CX contains the most significant part). On return, DX:AX contains the new location of the pointer (DX contains the most significant part).

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

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 Deny Read or Deny Write 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 mode.

On Entry	Register Contents
AH	43H
DS:DX	Pointer to an ASCIIZ path name
CX	Attribute
AL	Function code

On Return	Register Contents
AX	Error codes if carry flag set
CX	The file's current attribute; if carry flag not set and getting the attribute

Remarks: 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 then the file's attribute will be set to the attribute in CX. See "DOS Disk Directory" on page 5-10 for the attribute byte description. If AL is 00H then the file's current attribute is returned in CX.

43H**Change File Mode (CHMOD)**

Error codes are returned in AX. Issue function call 59H “Get Extended Error” for additional information about the error class, suggested action, and locus. Refer to “Error Return Information” on page 6-36 and “Extended Error Codes” on page 6-40 for more information on the codes returned from function call 59H.

Note: Attributes 08H and 10H cannot be changed using CHMOD. If they are used to change a file’s mode, an error code is returned.

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 (IOCTL)

Purpose: Sets or gets device information associated with open device handles, or send/receive control strings to the device handle.

On Entry	Register Contents
AH	44H
DS:DX	Data or buffer
CX	Number of bytes to read or write
BX	File handle
BL	Drive number (0=default, 1=A, etc.)
AL	Function value

On Return	Register Contents
AX	Number of bytes transferred if carry flag not set Error codes if carry flag set or if AX = 0FFH

Remarks: On entry, AL contains the function value. BX contains the file handle. On return, AX contains the number of bytes transferred for functions 2, 3, 4, and 5 or status (00H = not ready, FFH = ready) for functions 6 and 7, or an error code.

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action,

44H**I/O Control for Devices (IOCTL)**

and locus. Refer to “Error Return Information” on page 6-36 and “Extended Error Codes” on page 6-40 for more information on the codes returned from function call 59H.

The following function values are allowed in AL:

AL = 00H Get device information (returned in DX).

AL = 01H Set device information (determined by DX). Currently, DH must be zero for this call.

AL = 02H Read CX number of bytes into DS:DX from device control channel.

AL = 03H Write CX number of bytes from DS:DX to device control channel.

AL = 04H Same as 2, but use drive number in BL (0 = default, 1 = A, etc.)

AL = 05H Same as 3, but use drive number in BL (0 = default, 1 = A, etc.).

AL = 06H Get input status.

AL = 07H Get output status.

AL = 08H Is a particular block device changeable?

AL = 09H Is a logical device local or remote?

AL = 0AH Is a handle local or remote?

AL = 0BH Change sharing retry count.

IOCTL can be used to get information about device channels. You can make calls on regular files, but

44H

I/O Control for Devices (IOCTL)

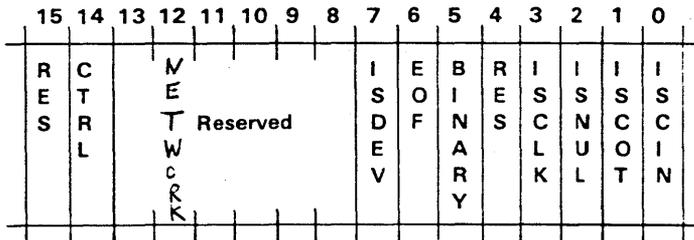
only function values 0, 6, and 7 are defined in that case. All other calls return an "Invalid Function" error.

Function values 00H to 08H are not supported on network devices. Function value 0BH requires the file sharing command to be loaded (SHARE).

Calls AL=00H and AL=01H.

BIT

The bits of DX are defined as follows:



ISDEV = 1 if this channel is a device.

= 0 if this channel is a disk file (bits 8-15
= 0 in this case).

If ISDEV = 1

EOF = 0 if end-of-file on input.

BIN = 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.

44H**I/O Control for Devices (IOCTL)**

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 zero for call AL=01H.

44H

I/O Control for Devices (IOCTL)

Calls AL=02H, AL=03H, AL=04H, AL=05H

These four calls allow arbitrary control strings to be sent or received from a device. The Call syntax is the same as the Read and Write calls, except for calls 4 and 5, which accept a drive number in BL instead of a handle in BX. An "Invalid Function" error is returned if the CTRL bit is zero. An "Access-Denied" code is returned by calls 04H and 05H if the drive is invalid.

Calls AL=06H and AL=07H

These calls allow you to check if a file handle is ready for input or output. If used for a file, AL always returns FFH 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 zero for not ready.

Call AL=08H (DOS 3.00 and 3.10)

This call allows you to determine if a device can support removable media. If the value returned in AX is 0, then the device is removable. If the value is 1, then the device is fixed. The drive number should be placed in BL. If the value in BL is invalid, then error code 0FH is returned. For network devices, the error Invalid function is returned.

44H**I/O Control for Devices (IOCTL)**

Call AL=09H (DOS 3.10)

This call allows you to determine if a logical device is associated with a network directory. 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 on local devices is the attribute word from the device header. On remote devices, bit 12 (1000H) is set. The other bits in DX are reserved. If disk redirection is paused, the function returns the attribute word for the local device and bit 12 is not set.

IMPORTANT: Do not write code that tests bit 12. Applications should be written so they are independent of the location (local or remote) of block devices.

Call AL=0AH (3.10)

This call allows you to determine if a handle is for a local device or a remote device across the network. The value returned in DX is the attribute word from the device header. For remote devices, it is bit 15 (8000H). The handle should be placed in BX.

IMPORTANT: Do not write code that tests bit 15. Applications should be written so they are independent of the location (local or remote) handles.

44H

I/O Control for Devices (IOCTL)

Call `AL=0BH` (DOS 3.00 and 3.10) All sharing and lock conflicts are automatically retried a number of times before they are returned as a DOS 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 never issued, DOS 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.

45H

Duplicate a File Handle (DUP)

Purpose: Returns a new file handle for an open file that refers to the same file at the same position.

On Entry	Register Contents
AH	45H
BX	File handle

On Return	Register Contents
AX	New file handle if carry flag not set Error codes if carry flag set

Remarks: 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 locus. Refer to “Error Return Information” on page 6-36 and “Extended Error Codes” on page 6-40 for more information on the codes returned from function call 59H.

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.

On Entry	Register Contents
AH	46H
BX	Existing file handle <i>old</i>
CX	Second file handle <i>new</i>

On Return	Register Contents
AX	Error codes if carry flag set None if carry flag not set

Remarks: 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, then 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

47H

Get Current Directory

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.

On Entry	Register Contents
AH	47H
DS:SI	Pointer to a 64 byte user memory area
DL	Drive number (0=default, 1=A, etc.)

On Return	Register Contents
DS:SI	Filled out with full path name from the root if carry is not set
AX	Error codes if carry flag is set

Remarks: 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

48H

Allocate Memory

Purpose: Allocates the requested number of paragraphs of memory.

On Entry	Register Contents
AH	48H
BX	Number of paragraphs of memory requested

On Return	Register Contents
AX:0	Points to the allocated memory block
AX	Error codes if carry set
BX	Size of the largest block of memory available (in paragraphs) if the allocation fails

Remarks: 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

49H

Free Allocated Memory

Purpose: Frees the specified allocated memory.

On Entry	Register Contents
AH	49H
ES	Segment of the block to be returned

On Return	Register Contents
AX	Error codes if carry flag set NONE if carry flag not set

Remarks: 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

4AH

Modify Allocated Memory Blocks (SETBLOCK)

Purpose: Modifies allocated memory blocks to contain the new specified block size.

On Entry	Register Contents
AH	4AH
ES	Segment of the block
BX	Contains the new requested block size in paragraphs

On Return	Register Contents
AX	Error codes if carry flag set None if carry flag not set
BX	Maximum poolsize possible if the call fails on a "grow request" if carry flag is set

Remarks: DOS attempts to "grow" or "shrink" the specified block.

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

4BH**Load or Execute a Program (EXEC)**

Purpose: Allows a program to load another program into memory and optionally begins execution of it.

On Entry	Register Contents
AH	4BH
DS:DX	Points to the ASCIIZ string with the drive, path, and filename to be loaded
ES:BX	Points to a parameter block for the load
AL	Function value (see description)

On Return	Register Contents
AX	Error codes if carry flag set NONE if carry flag not set

Remarks: Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

4BH

Load or Execute a Program (EXEC)

The following function values are allowed in AL:

Function Value	Description
00H	<p>Load and execute the program. A program segment prefix is established for the program; and the terminate and control-break addresses are set to the instruction after the EXEC system call.</p> <p>Note: When control is returned, all registers are changed, including the stack. You must restore SS, SP, and any other required registers before proceeding.</p>
03H	<p>Load, do not create the program segment prefix, and do not begin execution. This is useful in loading program overlays.</p>

4BH**Load or Execute a Program (EXEC)**

For each of these values, the block pointed to by ES:BX has the following format:

AL = 00H Load/execute program

WORD segment address of environment string to be passed
DWORD pointer to command line to be placed at PSP+80H
DWORD points to default FCB to be passed at PSP+5CH
DWORD pointer to default FCB to be passed at PSP+6CH

Note: The DWORD pointers are in offset segment form.

AL = 03H Load overlay

WORD segment address where file will be loaded
WORD relocation factor to be applied to the image

4BH

Load or Execute a Program (EXEC)

All open files of a process are duplicated in the newly created process after an EXEC, except if 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 32K bytes total) that convey 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 zero

Typically the environment strings have the form:

parameter=value

Following the byte of zero 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 zero value of the environment address causes the newly created process to inherit the parent's environment

4BH**Load or Execute a Program (EXEC)**

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 “Error Return Information” on page 6-36 and “Extended Error Codes” on page 6-40 for more information on the codes returned.

Notes:

1. 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.
2. The EXEC call uses the loader portion of COMMAND.COM to perform the loading.

4CH

Terminate a Process (EXIT)

Purpose: Terminates the current process and transfers control to the invoking process.

On Entry	Register Contents
AH	4CH
AL	Return code

On Return	Register Contents
	NONE

Remarks: 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.

On Entry	Register Contents
AH	4DH

On Return	Register Contents
AX	Return code

Remarks: The low byte of the exit code contains the information sent by the exiting routine. The high byte of the exit code can contain:

- 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

4EH

Find First Matching File (FIND FIRST)

Purpose: Finds the first filename that matches the specified file specification.

On Entry	Register Contents
AH	4EH
DS:DX	Pointer to an ASCIIZ string containing the drive, path, and filename of the file to be found
CX	Attribute used in searching for the file

On Return	Register Contents
AX	Error codes if carry flag set

Remarks: The filename in DS:DX can contain global filename characters. The ASCIIZ string cannot contain a network path. See function call 11H 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

4EH**Find First Matching File (FIND FIRST)**

If a file is found that matches the specified drive, path, and filename and attribute, the current DTA is filled in as follows:

21 bytes - reserved for DOS use on subsequent find next calls

1 byte - file's attribute

2 bytes - file's time

2 bytes - file's date

2 bytes - low word of file size

2 bytes - high word of file size

13 bytes - name and extension of file found, followed by a byte of zeros. All blanks are removed from the name and extension, and if an extension is present, it is preceded by a period. Thus, the name returned appears just as you would enter it as a command parameter, such as TREE.COM followed by a byte of zeros.

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.

On Entry	Register Contents
AH	4FH
DTA	Contains the information from a previous Find First or Find Next call (4EH, 4FH)

On Return	Register Contents
AX	Error codes if carry flag set

Remarks: If a matching file is found, the DTA is set as described in call 4EH. 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

54H

Get Verify Setting

Purpose: Returns the value of the verify flag.

On Entry	Register Contents
AH	54H

On Return	Register Contents
AL	Current verify flag value 00H, if verify is off 01H, if verify is on

Remarks: 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.

56H

Rename a File

Purpose: Renames the specified file.

On Entry	Register Contents
AH	56H
DS:DX	Pointer to an ASCIIZ string containing the drive, path, and filename of the file to be renamed
ES:DI	Pointer to an ASCIIZ string containing the new path and filename

On Return	Register Contents
AX	Error codes if carry flag set NONE if carry flag not set

Remarks: If a drive is used in the ASCIIZ string, it must be the same as the drive specified or implied in the first string. The directory paths need not be the same, allowing a file to be moved to another directory and renamed in the process. Global filename characters are not allowed in the filename.

56H

Rename a File

Error codes are returned in AX. Issue function call 59H “Get Extended Error” for additional information about the error class, suggested action, and locus. Refer to “Error Return Information” on page 6-36 and “Extended Error Codes” on page 6-40 for more information on the codes returned from function call 59H.

Network Access Rights: Requires Create access rights.

57H

Get/Set a File's Date and Time

Purpose: Gets or sets a file's date and time.

On Entry	Register Contents
AH	57H
AL	00H, get date and time 01H, set date and time
BX	File handle
CX	Time to be set if AL = 01H
DX	Date to be set if AL = 01H

On Return	Register Contents
AX	Error codes if carry flag set
DX	If getting date, date from the handle's internal table
CX	If getting time, time from the handle's internal table

Remarks: The date and time formats are the same as those for the directory entry described in Chapter 5 of this book, except that when passed in registers, the bytes are reversed (that is, DH contains the *low* order portion of the date, etc.).

57H**Get Set a File's Date and Time**

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

59H (DOS 3.00 and 3.10)

Get Extended Error

Purpose: Returns additional error information, such as the error class, locus, and recommended action.

On Entry	Register Contents
AH	59H
BX	0000H (version, 0 for 3.00 and 3.10)

On Return	Register Contents
AX	Extended error
BH	Error class
BL	Suggested action
CH	Locus

Remarks: This function call returns the error class, locus, 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 (DOS 3.00 and 3.10)

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:

1. Load up registers.
2. Issue interrupt 21H.
3. Continue operation, if carry not set.
4. Disregard the error code and issue function call 59H to obtain additional information.
5. 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:

1. Load up registers.
2. Issue interrupt 21H.
3. Continue operation, if error is not reported in AL.
4. Disregard the error code and issue function call 59H to obtain additional information.
5. Use the action in BL to determine the suggested action to take.

5AH (DOS 3.00 and 3.10)

Create Unique File

Purpose: Generates a unique filename, and creates that file in the specified directory.

On Entry	Register Contents
AH	5AH
DS:DX	Pointer to ASCIIZ path ending with a backslash (\)
CX	Attribute

On Return	Register Contents
AX	Error codes if carry flag is set
DS:DX	ASCIIZ path with the filename of the new file appended

Remarks: On entry, AH contains 5AH. If no error has occurred, then the file is opened in compatibility mode with Read/Write access, 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.

5AH (DOS 3.00 and 3.10) Create Unique File

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 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.

5BH (DOS 3.00 and 3.10)

Create New File

Purpose: Creates a new file.

On Entry	Register Contents
AH	5BH
DS:DX	Pointer an ASCIIZ path name
CX	File attributes

On Return	Register Contents
AX	Error codes if carry flag set Handle if carry flag not set

Remarks: This function call is identical to function call 3CH (Create) with the exception that it will fail if the filename already exists. The file is created in compatibility mode for reading and writing.

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

Network Access Rights: Requires Create access rights.

5CH (DOS 3.00 and 3.10)

Lock/Unlock File Access

Purpose: Locks or unlocks a range of bytes in an opened file.

On Entry	Register Contents
AH	5CH
AL	00H, to lock 01H, to unlock
BX	File handle
CX	Offset high
DX	Offset low
SI	Length high
DI	Length low

On Return	Register Contents
AX	Error codes if carry flag is set

Remarks: The Lock/Unlock function calls should only be used when a file is opened using the Deny Read or Deny None sharing modes, or when the file is opened for read/write or write only access and Deny Write sharing mode. These modes do no local buffering of data when accessing files on a network disk.

5CH (DOS 3.00 and 3.10)

Lock/Unlock File Access

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 to get the error code in addition to the error class, locus, 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 aborted using INT 23H or INT 24H 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 attempting 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 or INT 24H should trap these and release the lock before exiting. The proper

5CH (DOS 3.00 and 3.10)

Lock/Unlock File Access

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.

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

5E00H (DOS 3.10)

Get Machine Name

Purpose: Returns the character identifier of the local computer.

On Entry	Register Contents
AX	5E00H
DS:DX	Pointer to the memory buffer where the ASCIIZ computer name is returned

On Return	Register Contents
DS:DX	Filled with the ASCIIZ computer name
CH	Name/number indicator flag 0=name not defined not 0=name/number defined
CL	NETBIOS name number for the name
AX	Error codes if carry flag is set

Remarks: 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 Network Program must be loaded for the function call to execute properly.

5E02H (DOS 3.10)

Set Printer Setup

Purpose: Specifies an initial string for printer files.

On Entry	Register Contents
AX	5E02H
BX	Redirection list index
CX	Length of setup string (maximum length is 64 bytes)
DS:SI	Pointer to printer setup buffer

On Return	Register Contents
AX	Error codes if carry flag is set

Remarks: The string specified is put in front of all files destined for a particular network printer. 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 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

5E03H (DOS 3.10)

Get Printer Setup

Purpose: Returns the printer setup string for printer files.

On Entry	Register Contents
AX	5E03H
BX	Redirection list index
ES:DI	Pointer to printer setup buffer (maximum length is 64 bytes)

On Return	Register Contents
AX	Error codes if carry flag is set
CX	Length of data returned
ES:DI	Filled with the printer setup string

Remarks: 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 that is used in function call 5F02H (Get Redirection List Entry). Error code 1 (Invalid function number) is returned if the IBM PC 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 locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

5F02H (DOS 3.10)

Get Redirection List Entry

Purpose: Returns nonlocal network assignments.

On Entry	Register Contents
AX	5F02H
BX	Redirection index (zero-based)
DS:DI	Pointer to a 128-byte buffer address of the local device name
ES:DI	Pointer to a 128-byte buffer address of network name

On Return	Register Contents
AX	Error codes if carry flag is set
BH	Device status flag Bit 0=0 if device is valid 0=1 if device is not valid Bits 1-7 are reserved
BL	Device type
CX	Stored parm value
DX	Destroyed
BP	Destroyed
DS:SI	ASCIIZ local device name
ES:DI	ASCIIZ network name

5F02H (DOS 3.10)

Get Redirection List Entry

Remarks: 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 incremented by 1 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 Network Program is not loaded.

If either disk or print redirection is paused, the function is not affected.

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

5F03H (DOS 3.10)

Redirect Device

Purpose: Causes a Redirector/Server connection to be made.

On Entry	Register Contents
AX	5F03H
BL	Device type 03 Printer device 04 File device
CX	Value to save for caller
DS:SI	Source ASCIIZ device name
ES:DI	Destination ASCIIZ network path with password

On Return	Register Contents
AX	Error codes if carry flag is set

Remarks: This call is the interface that 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 maintains for the programmer. For compatibility with the IBM PC Network Program, CX should be set to 0. Values other than 0 are reserved for the IBM PC Network Program. This word may be retrieved through function call 5F02H (Get Redirection List). All output destined for the

5F03H (DOS 3.10)

Redirect Device

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 (zero length) password is considered to be no password.

- If BL = 4, the source specifies a drive letter and colon ended with 00H, the destination specifies a network path ended with 00H, and the CX register has a word that DOS maintains for the programmer. For compatibility with the IBM PC Network Program, CX should be set to 00H. Values other than 00H are reserved for the IBM PC 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.

5F03H (DOS 3.10)

Redirect Device

The ASCIIZ password for access to the remote path should immediately follow the network string. A null (zero length) password ended with 00H is considered to be no password.

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

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 file device while disk redirection is paused, or if you try to redirect a printer while print redirection is paused.

5F04H (DOS 3.10)

Cancel Redirection

Purpose: Cancels a previous redirection.

On Entry	Register Contents
AX	5F04H
DS:SI	ASCIIZ device name or path

On Return	Register Contents
AX	Error codes if carry flag is set

Remarks: 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 terminated 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 ended with 00H and a password ended with 00H, then 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 Network Program is not loaded.

5F04H (DOS 3.10)

Cancel Redirection

Error codes are returned in AX. Issue function call 59H "Get Extended Error" for additional information about the error class, suggested action, and locus. Refer to "Error Return Information" on page 6-36 and "Extended Error Codes" on page 6-40 for more information on the codes returned from function call 59H.

62H (DOS 3.00 and 3.10)

Get Program Segment Prefix Address (PSP)

Purpose: Returns the program prefix address.

On Entry	Register Contents
AH	62H

On Return	Register Contents
BX	Segment address of the currently executing process

Remarks: The internal PSP address for the currently executing process is returned in BX.

Chapter 7. DOS Control Blocks and Work Areas

Contents

Introduction	7-3
DOS Memory Map	7-4
DOS Program Segment	7-6
Program Segment Prefix	7-10
File Control Block	7-12
Standard File Control Block	7-13
Extended File Control Block	7-16

Introduction

This chapter contains:

- A description of the locations and usage of the DOS memory map.
- A detailed description and diagram of the program segment prefix.
- A detailed description and diagram of the file control block (standard and extended).

DOS Memory Map

Location	Usage
0000:0000	Interrupt vector table
0040:0000	ROM communication area
0050:0000	DOS communication area
XXXX:0000	IBMBIO.COM – DOS interface to ROM I/O routines
XXXX:0000	IBMDOS.COM – DOS interrupt handlers, service routines (INT 21 functions)
	DOS 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
XXXX:0000	External command or utility – (.COM or .EXE file)
XXXX:0000	User stack for .COM files (256 bytes)
XXXX:0000	Transient portion of COMMAND.COM

Notes:

1. Memory map addresses are in segment:offset format. For example, 0070:0000 is absolute address 0700H.
2. The DOS Communication Area is used as follows:

0050:0000	Print screen status flag store
0	Print screen not active or successful print screen operation
1	Print screen in progress
255	Error encountered during print screen operation
0050:0001	Used by BASIC
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 BASIC
0050:0022—002F	Used by DOS 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 use.
3. User memory is allocated from the lowest end of available memory that will satisfy the request for memory.

DOS Program Segment

When you enter an external command, or call a program through the EXEC function call, DOS determines the lowest available address to use as the start of available memory for the program being started. This area is called the Program Segment.

At offset 0 within the Program Segment, DOS builds the Program Segment Prefix control block. EXEC loads the program at offset 100H and gives it control.

The program returns from EXEC 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, or by calling location 50H in the Program Segment Prefix with AH=00H or 4CH.

Note: It is the responsibility of all programs to ensure that the CS register contains the segment address of the Program Segment Prefix when terminating using any of these methods except call 4CH.

All of these methods result in returning to the program that issued the EXEC. During this returning 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 terminate address.

When a program receives control, the following conditions are in effect:

For all programs:

- 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 32K bytes) in the form:

NAME=parameter

Each string is terminated by a byte of zeros, and the entire set of strings is terminated by another byte of zeros. Following the byte of zeros that terminates the set of environment strings is a set of initial arguments passed to a program that contains a word count followed by an ASCIIZ string. The ASCIIZ string contains the drive, path, and *filename[.ext]* of the executable program.

Programs may use this area to determine where the program was loaded from. The environment built by the command processor (and passed to all programs it invokes) contains a COMSPEC=*string* at a minimum (the parameter on COMSPEC is the path used by DOS to locate COMMAND.COM on disk). The last PATH and PROMPT commands issued will also be in the environment, along with any environment strings entered through the SET command. See Chapter 7 of the *DOS Reference* for more information.

The environment that you are passed is actually a copy of the invoking process environment. If your application uses a “terminate and stay resident” concept, you should be aware that the copy of the environment passed to you is static. That is, it will not change even if subsequent SET, PATH, or PROMPT commands are issued.

- Offset 50H in the Program Segment Prefix contains code to invoke the DOS function dispatcher. Thus, by placing the desired function number in AH, a program can issue a long call to PSP+50H to invoke a DOS function, rather than issuing an interrupt type 21H.
- Disk transfer address (DTA) is set to 80H (default DTA in the Program Segment Prefix).
- File control blocks at 5CH and 6CH are formatted from the first two parameters entered when the command was invoked. Note that if either parameter contained a path name, then the corresponding FCB will contain only a valid drive number. The filename field will not be valid.
- 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 reflects the validity of 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)

For .EXE programs:

- DS and ES registers are set to point to the Program Segment Prefix.
- CS, IP, SS, and SP registers are set to the values passed by the Linker.

For .COM programs:

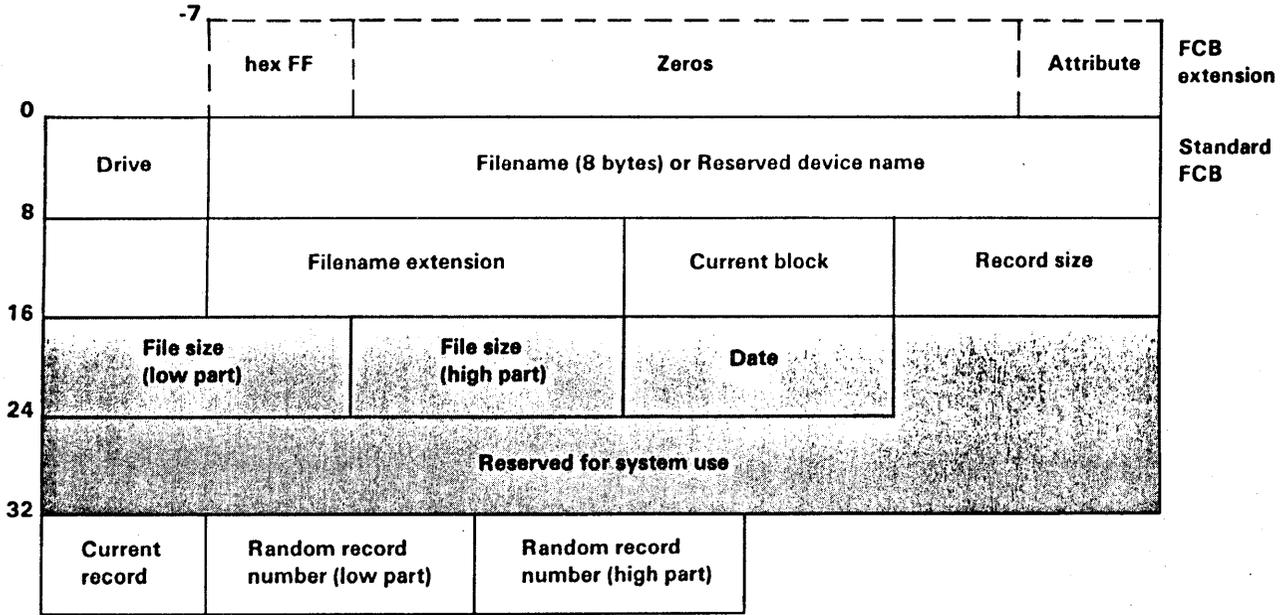
- All four segment registers contain the segment address of the initial allocation block, that starts with the Program Segment Prefix control block.
- All of user memory is allocated to the program. If the program wishes to invoke another program through the EXEC function call, it must first free some memory through the Setblock (4AH) function call, to provide space for the program being invoked.
- The Instruction Pointer (IP) is set to 100H.
- 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.
- A word of zeros is placed on the top of the stack.

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

Program Segment Prefix

0	1	2	3	4	5	6	7
INT 20H	Top of memory		Reserved	Long op code	Long call to DOS in segment		
8	9	A	B	C	D	E	F
cont segment of long call		Terminate address IP		Terminate address CS		Ctrl-break exit address IP	
10	11	12	13	14	15	16	17
Ctrl-break exit address CS		Critical error exit address IP		CS		Reserved	
18	19	2A	2B	2C	2D	2E	2F
RESERVED				Segment addresses of environment		Reserved	
30							4F
RESERVED							
50	51	52	53	54	55	56	57
DOS call		RESERVED					
58	59	5A	5B	5C	5D	5E	5F
Reserved				Standard FCB 1			
60	61	62	63	64	65	66	67
Standard FCB1 (cont)							
68	69	6A	6B	6C	6D	6E	6F
FCB1 (cont)				Standard FCB2			
70	71	72	73	74	75	76	77
Standard FCB2 (cont)							
78	79	7A	7B	7C	7D	7E	7F
Standard FCB2 (cont)							
80	81	82	83	84	85	86	87
parm length	Command parameters starting with leading blanks						
F8	F9	FA	FB	FC	FD	FE	FF

1. First segment of available memory is in segment (paragraph) form (for example, 1000H would represent 64K).
2. The word at offset 6 contains the number of bytes available in the segment.
3. Offset 2CH contains the segment address of the environment.
4. Programs must not alter any part of the PSP below offset 5CH.



(Offsets are in decimal)

Unshaded areas must be filled in by the using program.

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

Standard File Control Block

The standard file control block (FCB) is defined as follows, with the offsets in decimal:

Byte	Function
0	Drive number. For example, Before open: 0 - default drive 1 - drive A 2 - drive B etc. After open: 0 - drive A 1 - drive A 2 - drive B etc. 0 is replaced by the actual drive number during open.
1-8	Filename, left-justified with trailing blanks. If a reserved device name is placed here (such as LPT1), do not include the optional colon.
9-11	Filename extension, left-justified with trailing blanks (can be all blanks).
12-13	Current block number relative to the beginning of the file, starting with 0 (set to 0 by the open function call). A block consists of 128 records, each of the size specified in the logical record size field. The current block number is used with the current record field (below) for sequential reads and writes.

- 14-15 Logical record size in bytes. Set to 80H by the open function call. If this is not correct, you must set the value because DOS uses it to determine the proper locations in the file for all disk reads and writes.
- 16-19 File size in bytes. In this 2-word field, the first word is the low-order part of the size.
- 20-21 Date the file was created or last updated. The *mm/dd/yy* are mapped in the bits as follows:

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

where:

mm is 1-12

dd is 1-31

yy is 0-119 (1980-2099)

- 22-31 Reserved for system use.
- 32 Current relative record number (0-127) within the current block. (See above.) You must set this field before doing *sequential* read/write operations to the diskette. This field is not initialized by the open function call.
- 33-36 Relative record number relative to the beginning of the file, starting with 0. You must set this field before doing *random* read/write operations to the diskette. This field is not initialized by the open function call.

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

Notes:

1. An unopened FCB consists of the FCB prefix (if used), drive number, and filename/extensions properly filled in. An open FCB is one in which the remaining fields have been filled in by the Create or Open function calls.
2. Bytes 0-15 and 32-36 must be set by the user program. Bytes 16-31 are set by DOS and must not be changed by user programs.
3. 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.

Extended File Control Block

The extended File Control Block is used to create or search for files in the disk directory that have special attributes.

It adds a 7-byte prefix to the FCB, formatted as follows:

Byte	Function
FCB-7	Flag byte containing FFH to indicate an extended FCB.
FCB-6 to FCB-2	Reserved.
FCB-1	Attribute byte. See "DOS Disk Directory" on page 5-10 of this book for the attribute bit definitions. Also refer to function call 11H (search first) for details on using the attribute bits during directory searches. This function is present to allow applications to define their own files as <i>hidden</i> (and thereby exclude them from directory searches), and to allow selective directory searches.

Any reference in the DOS Function Calls (refer to Chapter 6 of the this book) to an FCB, whether opened or unopened, may use either a normal or 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.

Chapter 8. Executing Commands From Within an Application

Contents

Introduction	8-3
Invoking a Command Processor	8-3

Introduction

Application programs may invoke a secondary copy of the command processor. Your program may pass a DOS command as a parameter that the secondary command processor will execute as though it had been entered from the standard input device.

Invoking a Command Processor

The procedure is:

1. Assure that adequate free memory (17K bytes for DOS version 2.10 and 3.00; and 23K bytes for DOS version 3.10) exists to contain the second copy of the command processor and the command it is to execute. This is accomplished by executing function call 4AH to shrink memory allocated to that of your current requirements. Next, execute function call 48H with BX=FFFFH. This returns with the amount of memory available.

2. 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 assembly statement below would build the string to cause execution of a DISKCOPY command:

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

3. Use the EXEC function call (4BH, function value 0) to cause execution of the secondary copy of the command processor (the drive, directory, and name of the command processor can be gotten from the COMSPEC = *parameter* in the environment passed to you at PSP+2CH). Remember to set offset 2 of the EXEC control block to point to the string built above.

Chapter 9. Fixed Disk Information

Contents

Introduction	9-3
Fixed Disk Architecture	9-3
System Initialization	9-4
Boot Record/Partition Table	9-6
Fixed Disk Technical Information	9-8
Determining Fixed Disk Allocation	9-11

Introduction

The IBM Personal Computer Fixed Disk Support Architecture has been designed to meet the following objectives:

- Allow multiple operating systems to utilize the fixed disk without the need to backup/restore when changing operating systems.
- Allow a user-selected operating system to be started from the fixed disk.

Fixed Disk Architecture

The architecture is defined as follows:

- In order to *share* the fixed disk among operating systems, the disk may be logically divided into 1 to 4 partitions. The space within a given partition is contiguous, and can be dedicated to a specific operating system. Each operating system may “own” only one partition. The number and sizes of the partitions is user-selectable through a fixed disk utility program. The DOS utility is `FDISK.COM`. The partition information is kept in a partition table that is imbedded in the master fixed disk boot record on the first sector of the disk.
- Any operating system must consider its partition to be an entire disk, and must ensure that its functions and utilities do not access other partitions on the disk.

- Each partition can contain a boot record on its first sector, and any other programs or data that you choose—including a copy of an operating system. For example, the DOS FORMAT command may be used to format (and place a copy of DOS in) the DOS partition, in the same manner that a diskette is formatted. With the FDISK utility, you may designate a partition as “bootable” (active)—the master fixed disk boot record causes that partition’s boot record to receive control when the system is started or restarted.

System Initialization

The System initialization (or system boot) sequence is as follows:

1. System initialization first attempts to load an operating system from diskette drive A. If the drive is not ready or a read error occurs, it then attempts to read a master fixed disk boot record from the first sector of the first fixed disk on the system. If unsuccessful, or if no fixed disk is present, it invokes ROM BASIC.
2. If successful, the master fixed disk boot record is given control and it examines the partition table imbedded within it. If one of the entries indicates a “bootable” (active) partition, its boot record is read (from the partition’s first sector) and given control.
3. If none of the partitions is bootable, ROM BASIC is invoked.

4. If any of the boot indicators are invalid, or if more than one indicator is marked as bootable, the message **Invalid partition table** is displayed and the system enters an enabled loop. You may then insert a system diskette in drive A and use system reset to restart from diskette.
5. If the partition's boot record cannot be successfully read within five retries due to read errors, the message **Error loading operating system** appears and the system enters an enabled loop.
6. If the partition's boot record does not contain a valid "signature," the message **Missing operating system** appears, and the system enters an enabled loop. See "Boot Record Partition Table" on page 9-6 for complete information about the boot record.

Note: When changing the size or location of any partition, you must ensure that all existing data on the disk has been backed up (the partitioning process will "lose track" of the previous partition boundaries.)

Boot Record/Partition Table

A fixed disk boot record must be written on the first sector of all fixed disks, and contains:

1. Code to load and give control to the boot record for one of four possible operating systems.
2. A partition table at the end of the boot record. Each table entry is 16 bytes long, and contains the starting and ending cylinder, sector, and head for each of four possible partitions, as well as the number of sectors preceding the partition and the number of sectors occupied by the partition. The "boot indicator" byte is used by the boot record to determine if one of the partitions contains a loadable operating system. FDISK initialization utilities mark a user-selected partition as "bootable" by placing a value of 80H in the corresponding partition's boot indicator (setting all other partitions' indicators to 0 at the same time). The presence of the 80H tells the standard boot routine to load the sector whose location is contained in the following 3 bytes. That sector is the actual boot record for the selected operating system, and it is responsible for the remainder of the system's loading process (as it is from diskette). All boot records are loaded at absolute address 0:7C00.

The partition table with its offsets into the boot record is:

Offs Purpose	Head	Sector	Cylinder	
1BE Partition 1 begin	boot ind	H	S	CYL
1C2 Partition 1 end	syst ind	H	S	CYL
1C6 Partition 1 rel sect	Low word		High word	
1CA Partition 1 # sects	Low word		High word	
1CE Partition 2 begin	boot ind	H	S	CYL
ID2 Partition 2 end	syst ind	H	S	CYL
ID6 Partition 2 rel sect	Low word		High word	
IDA Partition 2 # sects	Low word		High word	
DE Partition 3 begin	boot ind	H	S	CYL
E2 Partition 3 end	syst ind	H	S	CYL
E6 Partition 3 rel sect	Low word		High word	
EA Partition 3 # sects	Low word		High word	
EE Partition 4 begin	boot ind	H	S	CYL
F2 Partition 4 end	syst ind	H	S	CYL
F6 Partition 4 rel sect	Low word		High word	
FA Partition 4 # sects	Low word		High word	
FE Signature	hex 55	hex AA		

Fixed Disk Technical Information

Boot Indicator (Boot Ind): The boot indicator byte must contain 0 for a non-bootable partition, or 80H for a bootable partition. Only one partition can be marked bootable.

System Indicator (Sys Ind): The “syst ind” field contains an indicator of the operating system that “owns” the partition.

The system indicators are:

00H - unknown (unspecified)

01H - DOS 12-bit FAT

04H - DOS 16-bit FAT

Cylinder (CYL) and Sector (S): The 1-byte fields labelled CYL contain the low-order 8 bits of the cylinder number—the high order 2 bits are in the high order 2 bits of the S (sector) field. This corresponds with ROM BIOS interrupt 13H (Disk I/O) requirements, to allow for a 10-bit cylinder number.

The fields are ordered in such a manner that only two MOV instructions are required to properly set up the DX and CX registers for a ROM BIOS call to load the appropriate boot record (fixed disk booting is only possible from the first fixed disk on a system, whose BIOS drive number (80H) corresponds to the boot indicator byte).

All partitions are allocated in cylinder multiples and begin on sector 1, head 0.

EXCEPTION: The partition that is allocated at the beginning of the disk starts at sector 2, to account for the disk's master boot record.

Relative Sector (Rel Sect): The number of sectors preceding each partition on the disk is kept in the 4-byte field labelled “rel sect.” This value is obtained by counting the sectors beginning with cylinder 0, sector 1, head 0 of the disk, and incrementing the sector, head, and then track values up to the beginning of the partition. Thus, if the disk has 17 sectors per track and 4 heads, and the second partition begins at cylinder 1, sector 1, head 0, the partition’s starting relative sector is 68 (decimal)—there were 17 sectors on each of 4 heads on 1 track allocated ahead of it. The field is stored with the least significant word first.

Number of Sectors (# Sects): The number of sectors allocated to the partition is kept in the “# of sects” field. This is a 4-byte field stored least significant word first.

Signature: The last 2 bytes of the boot record (55AAH) are used as a signature to identify a valid boot record. Both this record and the partition boot records are required to contain the signature at offset 1FEH.

The master disk boot record invokes ROM BASIC if no indicator byte reflects a “bootable” system.

When a partition’s boot record is given control, it is passed its partition table entry address in the DS:SI registers.

System programmers designing a utility to initialize/manage a fixed disk must provide the following functions at a minimum:

1. Write the master disk boot record/partition table to the disk’s first sector to initialize it.
2. Perform partitioning of the disk—that is, create or update partition table information (all fields for the partition) when the user wishes to create a partition. This may be limited to creating a partition for only one type of operating system, but must allow repartitioning the entire disk, or adding a partition without interfering with existing partitions (user’s choice).
3. Provide a means for marking a user-specified partition as bootable, and resetting the bootable indicator bytes for all other partitions at the same time.
4. Such utilities should not change or move any partition information that belongs to another operating system.

Determining Fixed Disk Allocation

DOS determines disk allocation using the following formula:

$$SPF = \frac{TS - RS - \frac{D * BPD}{BPS}}{CF + \frac{BPS * SPC}{BPC}}$$

The parameters are:

- TS* The count of the total sectors on the disk.
- RS* The number of sectors at the beginning of the disk that are reserved for the boot record. DOS reserves 1 sector.
- D* The number of directory entries in the root directory. Refer to "DOS Disk Directory" on page 5-10 for more information.
- BPD* The number of bytes per directory entry. BPD is always 32.
- BPS* The number of bytes per logical sector. Typically, BPS is 512, but you can specify a different value using **VDISK**.
- CF* The number of FATs per disk. For most disks CF is 2. For **VDISK** CF is 1.
- SPF* The number of sectors per FAT. The maximum value for SPF is 64.
- SPC* The number of sectors per allocation unit.
- BPC* The number of bytes per FAT entry. BPC is 1.5 for 12-bit FATs and 2 for 16-bit FATs.

Chapter 10. EXE File Structure and Loading

Contents

Introduction	10-3
.EXE File Structure	10-3
The Relocation Table	10-5

10-2

Introduction

This chapter contains information on:

- The .EXE file structure
- The relocation table

.EXE File Structure

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

- Control and relocation information
- The load module itself

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

The header is formatted as follows:

Hex Offset	Contents
00-01	4DH, 5AH—This is the Link program's <i>signature</i> to mark the file as a valid .EXE file.
02-03	Length of image mod 512 (remainder after dividing the load module image size by 512).
04-05	Size of the file in 512-byte increments (<i>pages</i>), including the header.
06-07	Number of relocation table items .
08-09	Size of the header in 16-byte increments (<i>paragraphs</i>). This is used to locate the beginning of the load module in the file.
0A-0B	Minimum number of 16-byte paragraphs required above the end of the loaded program.
0C-0D	Maximum number of 16-byte paragraphs required above the end of the loaded program.
0E-0F	Displacement in paragraphs of stack segment within load module.
10-11	Offset to be in the SP register when the module is given control.
12-13	Word checksum—negative sum of all the words in the file, ignoring overflow.
14-15	Offset to be in the IP register when the module is given control.
16-17	Displacement in paragraphs of code segment within load module.
18-19	Displacement in bytes of the first relocation item within the file.
1A-1B	Overlay number (0 for resident part of the program).

Note: Use the value at hex offset 18–19 to located the first entry in the relocation table.

The Relocation Table

The word at 18H locates the first entry in the relocation table. The relocation table is made up of a variable number of relocation items. The number of items is contained at offset 06-07. The relocation item contains two fields—a 2-byte offset value, followed by a 2-byte segment value. These two fields represent the displacement into the load module of a word which requires modification before the module is given control. This process is called *relocation* and is accomplished as follows:

1. A program segment prefix is built following the resident portion of the program that is performing the load operation.
2. The formatted part of the header is read into memory (it's size is at offset 08-09).
3. The load module size is determined by subtracting the header size from the file size. Offsets 04-05 and 08-09 can be used for this calculation. The actual size is downward adjusted based on the contents of offsets 02-03. Note that all files created by Link programs prior to version 1.10 *always* placed a value of 4 at that location, regardless of actual program size. Therefore, we recommend that this field be ignored if it contains a value of 4. Based on the setting of the high/low loader switch, an appropriate segment is determined at which to load the load module. This segment is called the *start segment*.
4. The load module is read into memory beginning at the start segment.

Note: The relocation table is an unordered list of relocation items. The first relocation item is the one that has the lowest offset in the file.

5. The relocation table items are read into a work area (one or more at a time).
6. Each relocation table item segment value is added to the start segment value. This calculated segment, in conjunction with the relocation item offset value, points to a word in the load module to which is added the start segment value. The result is placed back into the word in the load module.
7. Once all relocation items have been processed, the SS and SP registers are set from the values in the header and the start segment value is added to SS. The ES and DS registers are set to the segment address of the program segment prefix. The start segment value is added to the header CS register value. The result, along with the header IP value, is used to give the module control.

Chapter 11. DOS Memory Management

Contents

Introduction	11-3
Control Block	11-3

Introduction

DOS keeps track of allocated and available memory blocks, and provides three function calls for application programs to communicate their memory needs to DOS. These calls are 48H to allocate a memory block, 49H to free a previously allocated memory block, and 4AH (SETBLOCK) to change the size of an allocated memory block.

Control Block

DOS manages memory as follows:

DOS builds a control block for each block of memory, whether free or allocated. For example, if a program issues an “allocate,” DOS locates a block of free memory that satisfies the request, and will “carve” the requested memory out of that block. The requesting program is passed the location of the first byte of the block that was allocated for it—a memory management control block, describing the allocated block, has been built for the allocated block and a second memory management control block describes the amount of space left in the original free block of memory. When you do a setblock to shrink an allocated block, DOS builds a memory management control block for the area being freed, and adds it to the chain of control blocks. Thus, any program that changes memory that is not allocated to it, stands a chance of destroying a DOS memory management control block. This causes unpredictable results that don’t show up until an activity is performed where DOS uses its chain of control blocks (the normal result is a memory allocation error, for which the only corrective action is to restart the system).

When a program (command or application program) is to be loaded, DOS uses the EXEC function call (4BH)

to perform the loading. This is the same function call that is available to application programs for loading other programs. This function call has 2 options,

- Function 0, to load and execute a program (this is what the command processor uses to load and execute external commands).
- Function 3, to load an overlay (program) without executing it.

Although both functions perform their loading in the same way (relocation is performed for .EXE files), their handling of memory management is different.

Function 0: For function 0 to load and execute a program, EXEC first allocates the largest available block of memory (the new program's PSP will be at offset 0 in that memory block). Then EXEC loads the program. Thus, in most cases, the new program "owns" all of the memory from its PSP to the highest end of memory, including the memory occupied by the transient part of COMMAND.COM. If the program were to issue its own EXEC function call to load and execute another program, the request would fail because no available memory exists to load the new program into.

Note: For .EXE programs, the amount of memory allocated is the size of the program's memory image plus the value in the MAX ALLOC field of the file's header (offset 0CH, if that much memory is available. If not, EXEC allocates the size of the program's memory image plus the value in the MIN ALLOC field in the header (offset 0AH). These fields are set by the Linker.

A well-behaved program uses the SETBLOCK function call when it receives control, to shrink its allocated memory block down to the size it really needs. A .COM program should remember to set up its own stack before doing the SETBLOCK, since it is likely that the default stack supplied by DOS lies in the area

of memory being freed. This frees unneeded memory, which can then be used for loading subsequent programs.

If the program requires additional memory during processing, it can obtain the memory using the `allocate` function call and later free it using the `free memory` function call.

When a program loaded using `EXEC` function 0 exits, its initial allocation block (the block beginning with its `PSP`) is automatically freed before the calling program regains control. It is the responsibility of all programs to free any memory they allocate, before exiting to the calling program.

Function 3: For function 3, to load an overlay, no `PSP` is built, and `EXEC` assumes the calling program has already allocated memory to load the new program into—it will *not* allocate memory for it. Thus, the calling program should either allow for the loading of overlays when it determines the amount of memory to keep when issuing the `SETBLOCK` call, or should initially free as much memory as possible. The calling program should then allocate a block (based on the size of the program to be loaded) to hold the program that will be loaded using the “load overlay” call. Note that “load overlay” does not check to see if the calling program actually owns the memory block it has been instructed to load into—it assumes the calling program has followed the rules. If the calling program does not own the memory into which the overlay is being loaded, there is a chance that the program being loaded will overlay one of the control blocks that `DOS` uses to keep track of memory blocks.

Programs loaded using function 3 should *not* issue any `SETBLOCK` calls, since they don't own the memory they are operating in (the memory is owned by the *calling* program).

Because programs loaded using function 3 are given control directly by (and return control directly to) the calling program with no DOS intervention, no memory is automatically freed when the called program exits—it is up to the calling program to determine the disposition of the memory that had been occupied by the exiting program. Note that if the exiting program had itself allocated any memory, it is responsible for freeing that memory before exiting.

Index

Special Characters

.COM filename extension 1-6,
1-7
.COM programs 7-9
.EXE file structure 10-3
.EXE filename extension 1-6,
1-7, 7-6
.EXE files, load 10-3
.EXE programs 7-9
/S option 5-13

A

abort program 1-10
absolute disk, interrupt
 read 6-22
 write 6-23
absolute track/sector,
 calculate 5-13
access rights, network 6-45
actions, error recovery 6-43
address terminate
 interrupt 6-13
address, program segment
 prefix 6-178
AH register 6-32
allocate memory 6-142
allocated memory blocks,
 modify 6-144
allocated memory, free 6-143

allocating diskette space 5-5
allocating file space 4-14
allocation table
 information 6-76
allocation table information,
 specific device 6-77
allocation, diskette 5-4
application, executing
 commands within your 8-3
area for DOS 5-4
ASCII codes, extended 6-11
ASCII mode 4-9
ASCII mode, file I/O 4-11
ASCII string 6-44
attribute byte 7-16
attribute field 2-7
attribute field bits
 clock device 2-8
 device type 2-7
 format 2-8
 IOCTL 2-7
 NUL 2-9
 removable media 2-8
 standard input 2-9
 standard output 2-9
attribute, file 5-11
AUTOEXEC file 1-5
Auxiliary Asynchronous
 Communications
 Adapter 6-52
auxiliary input 6-52
auxiliary output 6-52, 6-53
available functions, DOS 1-8
AX register 6-16, 6-47

B

- base pointer 6-10
- base register 6-9
- batch file processor 1-6
- binary mode 4-9
- binary mode, file I/O 4-10
- BIOS 6-22
- BIOS interface module 1-4
- BIOS parameter block 2-26
- bit fields 6-114
- block devices 2-5
- block devices, installing 2-12
- block number, current 7-13
- block read, random 6-78
- block write, random 6-79, 6-85
- blocking/de-blocking, data 1-5
- boot record 1-4, 5-4
- boot sector format 2-28
- BP register 6-16
- BPB, BIOS parameter block 2-12
- buffer 1-9
- buffered standard input 6-59
- buffers, file 6-12
- BUILD BPB function call parameter 2-26
- built-in functions 1-4
- busy bit 2-16
- BX register 6-16
- byte, attribute 7-16
- byte, flag 7-16
- track/sector 5-13
- calls, function 6-32
- cancel redirection 6-176
- change current directory 6-108
- change file mode 6-130
- character devices 2-5
- character devices, installing 2-12
- check keyboard status 6-60
- check, ctrl-break 6-96
- checksum methodology 1-5
- CL register 6-47
- clear keyboard buffer 6-61
- clock device bit 2-8
- CLOCK\$ device 2-36
- close a file handle 6-122
- close file 6-65
- CLOSE function call parameter 2-34
- cluster number, relative 5-13
- cluster, calculate 5-8
- cluster, locate next 5-7, 5-8
- cluster, starting 5-13
- clusters 5-15
- code segment 6-10
- COMMAND.COM 5-13, 7-6
- command code 2-15
- command processor 1-5
- command processor portions initialization 1-5 resident 1-5 transient 1-6
- communications adapter, auxiliary asynchronous 6-52
- compatibility mode 6-117
- components of DOS 1-4
- console I/O, direct 6-55

C

- calculate absolute cluster 5-8
- calculate absolute

- console input without
 - echo 6-57
- console/keyboard routines 1-8
- control blocks 7-3
- control screen cursor 3-3
- control sequences 3-3
- control, for device I/O 6-132
- count register 6-9
- country dependent
 - information 6-99, 6-101
- create a file 6-109
- create file 6-72
- create subdirectory 6-106
- create unique file 6-162
- creating a device driver 2-10
- critical error handler 1-5
- critical error handler
 - vector 6-14
- CS register 6-12, 6-16, 7-6, 7-9
- Ctrl-break check 6-96
- Ctrl-Break handler 1-5
- Ctrl-Break exit address
 - interrupt 6-13
- current block number 7-13
- current directory,
 - change 6-108
- current directory, get 6-141
- current disk 6-74
- current relative record
 - number 7-14
- cursor backward 3-8
- cursor control 3-6
- cursor control sequences
 - cursor backward 3-8
 - cursor down 3-7
 - cursor forward 3-8
 - cursor position 3-6
 - cursor position report 3-10
 - cursor up 3-7
 - device status report 3-10
 - erase in display 3-13
 - erase in line 3-13
 - horizontal position 3-9
 - keyboard key
 - reassignment 3-17
 - reset mode 3-16
 - restore cursor
 - position 3-12
 - save cursor position 3-12
 - set graphics rendition 3-15
 - set mode 3-16
 - vertical position 3-9
- cursor up 3-7
- CX register 6-16

D

- data area 5-14
- data blocking/de-blocking 1-5
- data register 6-9
- data segment 6-10
- date
 - get 6-88
 - set 6-89
- date file created or
 - updated 7-14
- de-blocking/blocking,
 - data 1-5
- defective tracks 5-13
- delete a file from a
 - directory 6-127
- delete file 6-69
- deny none mode 6-120
- deny read mode 6-119
- deny read/write mode 6-119
- destination index 6-10
- device driver functions
 - BUILD BPB 2-26
 - CLOSE 2-34
 - FLUSH 2-33
 - INIT 2-19
 - INPUT 2-29

- MEDIA CHECK 2-21
- MEDIA
 - DESCRIPTOR 2-23
 - NONDESTRUCTIVE
 - INPUT 2-31
 - OPEN 2-34
 - OUTPUT 2-29
 - REMOVABLE
 - MEDIA 2-35
 - STATUS 2-32
- device driver, creating 2-10
- device driver, sample
 - listing 2-36
- device drivers, DOS
 - clock\$ device 2-36
 - creating 2-10
 - device header 2-6
 - format 2-4
 - installing 2-11
 - request header 2-14
 - sample listing 2-36
 - status word 2-16
 - types 2-5
- device drivers, installing 2-11
- device field, next 2-6
- device header 2-6
- device header fields
 - attribute 2-7
 - interrupt routine 2-9
 - name/unit 2-9
 - next device header 2-6
 - strategy routine 2-9
- device status report 3-10
- device type bit 2-7
- device, I/O control 6-132
- device, read from 6-123
- device, write to 6-125
- devices, types of 2-5
- DI register 6-15, 6-16
- direct console I/O 6-55
- direct console input without
 - echo 6-56
- directory entries
 - file attribute 5-11
 - file creation date 5-13
 - file creation time 5-12
 - file extension 5-11
 - file size 5-13
 - filename 5-10
 - directory searches 5-11
 - directory, change 6-108
 - directory, get current 6-141
 - disk
 - current 6-74
 - error handling 1-5
 - errors 6-18
 - free space 6-98
 - read, absolute 6-22
 - reset 6-62
 - select 6-63
 - write, absolute 6-23
 - disk transfer address 7-7
 - disk transfer address, set 6-75
 - disk transfer area (DTA) 1-9
 - diskette
 - allocating space 5-5
 - allocation 5-3
 - defective tracks 5-13
 - directory 5-10
 - handling routines 1-8
 - display output 6-13, 6-51
 - done bit 2-16
 - DOS
 - area 5-4
 - available functions 1-8
 - control blocks 7-3
 - data area 5-14
 - disk allocation 5-3
 - diskette directory 5-10
 - flags 6-9
 - function calls 6-32
 - general registers 6-9
 - index registers 6-10
 - initialization 1-7
 - interrupts 6-12
 - memory map 7-3

- pointer 6-10
 - program segment 7-6
 - registers 6-9
 - segment registers 6-10
 - structure 1-4
 - technical information 1-3
 - work areas 7-3
 - DOS components
 - boot record 1-4
 - command processor 1-5
 - DOS program file 1-5
 - read only memory 1-4
 - DOS environment 7-7
 - DOS function calls, see function calls also
 - DOS interrupts, see interrupts
 - function request 6-13
 - DOS program file 1-5
 - DOS registers 6-9
 - DOS registers, see registers, DOS
 - DOS register 6-16, 7-9
 - DTA (disk transfer area) 1-9
 - duplicate a file handle 6-139
 - DX register 6-16
- }
- end-of-file mark 5-7
 - entries, search for 6-66
 - environment, DOS 7-7
 - erase control sequences
 - erase in display 3-13
 - erase in line 3-13
 - erase in display control
 - sequence 3-13
 - erase in line control
 - sequence 3-13
 - erasing control sequences 3-13
 - error bit 2-16
 - error classes 6-42
 - error codes
 - interrupt 24H 6-14
 - error codes, interrupt 2FH 6-27
 - error codes, status word 2-17
 - error handler 1-10
 - error handling
 - critical 1-5
 - disk 1-5
 - error return information 6-36
 - error trapping 1-10
 - errors, disk 6-18
 - ES register 6-16, 7-9
 - EXEC, load or execute a program 6-145
 - execute a program, EXEC 6-145
 - executing commands within an application 8-3
 - EXIT, terminate a process 6-150
 - extended ASCII codes 6-11
 - extended error codes 6-40
 - extended file control
 - block 7-16
 - extended function calls 4-3
 - extended, 59H 6-40
 - extension
 - .COM 1-6, 1-7
 - .EXE 1-6, 1-7, 7-6
 - external commands 1-6
 - extra segment 6-10
- ## F
- FAT (see File Allocation Table)
 - FCB 7-15
 - FCB function calls 4-3, 4-5

- FCB restrictions 4-12
- field name 2-9
- field, attribute 2-7
- file
 - attribute 5-11
 - change mode 6-130
 - close 6-65
 - create 6-72, 6-109
 - date created or updated 7-14
 - delete 6-69
 - find first matching
 - file 6-152
 - find next matching
 - file 6-154
 - hidden 5-10, 6-72, 7-16
 - move read/write
 - pointer 6-128
 - open 6-64, 6-110, 6-112
 - rename 6-73
 - size 6-80
 - system 7-16
- file access, lock/unlock 6-165
- File Allocation Table (FAT) 5-5
- file allocation table, how to use 5-8
- file buffers 6-12
- File Control Block (FCB) 7-12
- file control block function calls 4-3
- file control block, extended 7-16
- file handle 4-7, 6-46
- file handle, closing 6-122
- file handle, duplicate 6-139
- file handles 6-46
 - standard auxiliary device 4-8
 - standard error device 4-8
 - standard input device 4-8
 - standard output device 4-8
- standard printer device 4-8
- file I/O
 - ASCII mode 4-11
 - binary mode 4-10
- file management functions 4-3
- file sectors
- file size 7-14
- file structure, .EXE 10-3
- file, allocating space 4-14
- file, read from 6-123
- file, write to 6-125
- filename
 - in directory 5-11
 - in file control block 7-13
- filename extension
 - .COM 1-6, 1-7
 - .EXE 1-6, 1-7, 7-6
 - in directory 5-11
 - in file control block 7-13
 - separators 6-87
 - terminators 6-87
- filename, parse 6-86
- find first matching file 6-152
- FIND FIRST, find first matching file 6-152
- find next matching file 6-154
- FIND NEXT, find next matching file 6-154
- flag byte 7-16
- flags 6-9
- FLUSH function call
 - parameter 2-33
- format bit 2-8
- FORMAT command 5-10
- format, device drivers 2-4
- free allocated memory 6-143
- function call 31H 6-24
- function calls
 - function calls, INT21
 - allocate memory 6-142
 - allocation table information 6-76

- allocation table information
 - for specific device 6-77
- auxiliary input 6-52
- auxiliary output 6-53
- buffered keyboard
 - input 6-59
- cancel redirection 6-176
- change current
 - directory 6-108
- change file mode 6-130
- check standard input
 - status 6-60
- clear keyboard buffer and
 - invoke a keyboard function 6-61
- close a file handle 6-122
- close file 6-65
- console input without
 - echo 6-57
- create a file 6-109
- create file 6-72
- create new file 6-164
- create new program
 - segment 6-83
- create subdirectory 6-106
- create unique file 6-162
- ctrl-break check 6-96
- current disk 6-74
- delete a file from a
 - directory 6-127
- delete file 6-69
- direct console I/O 6-55
- direct console input without
 - echo 6-56
- disk reset 6-62
- display output 6-51
- duplicate a file
 - handle 6-139
- FCB function calls 4-5
- file size 6-80
- find first matching
 - file 6-152
- find next matching
 - file 6-154
- force a duplicate
 - handle 6-140
- free allocated
 - memory 6-143
- get a return code of a
 - subprocess 6-151
- get current directory 6-141
- get date 6-88
- get disk free space 6-98
- get disk transfer
 - address 6-93
- get DOS version
 - number 6-94
- get extended error 6-160
- get machine name 6-168
- get or set country
 - dependent information 6-101
- get printer setup 6-170
- get program segment prefix
 - address 6-178
- get redirection list
 - entry 6-171
- get time 6-90
- get vector 6-97
- get verify setting 6-155
- get/set file's date and
 - time 6-158
- handle function calls 4-6
- I/O control for
 - devices 6-132
- keyboard input 6-50
- load or execute a
 - program 6-145
- lock/unlock file
 - access 6-165
- modify allocated memory
 - blocks 6-144
- move file read/write
 - pointer 6-128
- open a file 6-110, 6-112

G

open file 6-64
 parse filename 6-86
 print string 6-58
 printer output 6-54
 program terminate 6-49
 random block read 6-84
 random block write 6-85
 random read 6-78
 random write 6-79
 read from a file or
 device 6-123
 redirect device 6-173
 remove subdirectory 6-107
 rename a file 6-156
 rename file 6-73
 return country dependent
 information 6-99
 search for first entry 6-66
 search for next entry 6-68
 select disk 6-63
 sequential read 6-70
 sequential write 6-71
 set date 6-89
 set disk transfer
 address 6-75
 set interrupt vector 6-82
 set printer setup 6-169
 set relative record
 field 6-81
 set time 6-91
 set/reset verify
 switch 6-92
 terminate a process 6-150
 terminate process and
 remain resident 6-95
 write to a file or
 device 6-125
 function codes, interrupt
 2FH 6-27
 function request interrupt 6-13
 functions, available DOS 1-8
 functions, built-in 1-4
 functions, device drivers 2-18

general registers 6-9
 get
 country dependent
 information 6-101
 current directory 6-141
 date 6-88
 disk free space 6-98
 disk transfer address 6-93
 DOS version number 6-94
 get machine name 6-168
 printer setup 6-170
 program segment prefix
 address 6-178
 redirection list entry 6-171
 time 6-90
 vector 6-97
 verify setting 6-155
 get a file's date and time 6-158
 get extended error, function
 call 6-160
 get or set country (DOS 3.00
 and 3.10) 6-101

H

handle
 duplicate 6-139
 file 6-46
 force a duplicate 6-140
 function calls 4-3
 restrictions on usage 4-13
 standard 4-8
 handle function calls 4-6

handle, file 6-46
 handle, force a duplicate 6-140
 header 10-3
 hidden files 5-10, 6-72, 7-16
 high memory 1-6
 high/low loader switch 10-5
 horizontal position 3-9

I

I/O control for devices 6-132
 IBMBIO.COM 1-7, 5-13
 IBMDOS.COM 1-7, 5-13
 index register 6-10
 INT function call
 parameter 2-19
 initialization portion of
 command processor 1-5
 initializing DOS 1-7
 INPUT function call
 parameter 2-29
 nput, auxiliary 6-52
 nstalling block devices 2-12
 nstalling character
 devices 2-12
 nstalling device drivers 2-11
 nstruction pointer 6-10
 NT 24H 1-10
 nterface module,
 IBMBIO.COM 1-4
 nternal command
 processors 1-6
 nterrupt routines 2-9
 nterrupt vectors 1-7
 nterrupt, set 6-82
 nterrupts, DOS
 absolute disk read 6-22
 absolute disk write 6-23
 critical error handler
 vector 6-15

ctrl-break exit address 6-13
 function request 6-13
 multiplex 6-26
 program terminate 6-12
 terminate address 6-13
 terminate but stay
 resident 6-24

INT21, function calls 6-32
 invoke keyboard function 6-61
 invoking DOS functions 6-47
 IOCTL 6-132
 IOCTL bit 2-7
 IP register 6-16, 7-9
 IRET 6-13

K

keyboard function,
 invoke 6-61
 keyboard input 6-50
 keyboard input, buffered 6-59
 keyboard reassignment 3-17
 keyboard status, check 6-60
 keys, reassign 3-17

L

linefeed 6-13
 load a program, EXEC 6-145
 loading .EXE files 10-3
 locate next cluster 5-7, 5-8
 lock file access 6-165
 logical record size 7-14
 logical sector numbers 6-22

M

MEDIA CHECK function call
 parameter 2-21
 media descriptor byte 2-23
 memory
 allocating 6-142
 freeing allocated 6-143
 high 1-6
 image file 2-4
 map, DOS 7-3
 modify blocks 6-144
 memory map
 mode of operation control
 sequences
 reset mode 3-16
 set graphics rendition 3-15
 set mode 3-16
 modify allocated memory
 blocks 6-144
 MOV instruction 5-8
 move file read/write
 pointer 6-128
 multiplex interrupt 6-26

N

name/unit field 2-9
 network access rights 6-45
 new file, creating 6-164
 next device 2-6
 next entry, search 6-68
 nondestructive input function
 call parameter 2-31
 NUL bit 2-9

O

open a file using
 handles 6-110, 6-112
 open file using FCBs 6-64
 OPEN function call
 parameter 2-34
 open mode 6-114
 output
 auxiliary 6-53
 display 6-51
 printer 6-54
 routines 1-8
 OUTPUT function call
 parameter 2-29
 output, display 6-13

P

parse filename 6-86
 pointers 6-10
 predefined handles 6-46
 print string 6-58
 printer output 6-54
 printer output routines 1-8
 printer setup 6-169, 6-170
 program segment
 create new 6-83
 DOS 7-6
 Program Segment Prefix 1-9,
 7-9, 7-10
 program terminate 6-49
 program terminate
 interrupt 6-12

R

- random block read 6-84
- random block write 6-85
- random read 6-78
- random record field, set 6-81
- random write 6-79
- read from a file or device 6-123
- read only 1-4
- read only memory (ROM) interface 1-4
- read, random 6-78
- read, random block 6-84
- read, sequential 6-70
- reassign keys 3-17
- record number, relative 7-14
- record size, logical 7-14
- redirect device 6-173
- redirection list entry 6-171
- registers, DOS
 - AH 6-9
 - AL 6-9
 - AX 6-9
 - base 6-9
 - base pointer 6-10
 - BH 6-9
 - BL 6-9
 - BX 6-9
 - CH 6-9
 - CL 6-9
 - code segment 6-10
 - count 6-9
 - CX 6-9
 - data 6-9
 - data segment 6-10
 - destination index 6-10
 - DH 6-9
 - DL 6-9
 - DX 6-9
 - extra segment 6-10
 - general registers 6-9
 - index 6-10
 - instruction pointer 6-10
 - pointers 6-10
 - segment register 6-10
 - stack index 6-10
 - stack pointer 6-10
 - stack segment 6-10
- relative cluster number 5-13
- relative record number 7-14
- relocation 10-5
- removable media bit 2-8
- removable media function call parameter 2-35
- remove subdirectory 6-107
- rename a file 6-156
- rename file 6-73
- request header 2-14
 - command code 2-15
 - status word 2-16
 - unit code 2-14
- reset mode, control sequence 3-16
- reset, disk 6-62
- reset, system 1-7
- resident portion of command processor 1-5
- restore cursor position 3-12
- restriction on FCB usage 4-12
- restriction on handle usage 4-13
- return country dependent information 6-99
- ROM (read only memory) 1-4
- ROM BIOS interface module 1-4
- ROM BIOS routine 6-52
- routines
 - console/keyboard 1-8
 - device 1-4
 - diskette handling 1-8
 - keyboard input 1-8
 - output 1-8
 - printer output 1-8

- ROM BIOS 6-52
 - time function 1-8
 - routines,
 - strategy/interrupt 2-9
- S**
- sample device driver
 - listing 2-36
 - save area, parameter 7-8
 - save cursor position 3-12
 - screen cursor control 3-6
 - search for entries 6-66
 - search, next entry 6-68
 - sector numbers, logical 6-22
 - segment address 7-7
 - segment register 6-10
 - segment, create new
 - program 6-83
 - segment, start 10-5
 - select disk 6-63
 - separators, filename 6-87
 - sequential read 6-70
 - sequential write 6-71
 - set
 - country dependent
 - information 6-101
 - date 6-89
 - interrupt 6-82
 - printer setup 6-169
 - random record field 6-81
 - time 6-91
 - verify switch 6-92
 - set a file's date and time 6-158
 - set disk transfer address 6-75
 - set graphics rendition, control
 - sequence 3-15
 - set mode, control
 - sequence 3-16
 - setup, printer 6-169, 6-170
 - sharing modes 6-117
 - SI register 6-16
 - single-drive system 6-63
 - size, file 6-80, 7-14
 - SP register 7-9
 - space allocation 5-3
 - specific device allocation table
 - information 6-77
 - SS register 7-9
 - stack index 6-10
 - stack pointer 6-10
 - stack segment 6-10
 - stack space 1-7
 - stack, user 6-16
 - standard file handles 4-8
 - standard input bit 2-9
 - standard output bit 2-9
 - start segment 10-5
 - starting cluster 5-13
 - static environment 7-7
 - STATUS function call
 - parameter 2-32
 - status word 2-16
 - status word bits
 - busy 2-16
 - done 2-16
 - error 2-16
 - error code 2-17
 - strategy routines 2-9
 - strings, ASCIIZ 6-44
 - structure, DOS 1-4
 - subdirectory, create 6-106
 - subdirectory, remove 6-107
 - switch, high/low loader 10-5
 - system file 7-16
 - system prompt 1-6
 - system reset 1-7

T

technical information,
 DOS 1-3
 terminate a process 6-150
 terminate address
 interrupt 6-13
 terminate but stay
 resident 1-5, 6-95
 terminate but stay resident
 interrupt 6-24
 terminate process and stay
 resident 6-95
 terminate program 6-49
 terminate program
 interrupt 6-13
 terminators, filename 6-87
 time
 get 6-90
 set 6-91
 time function routines 1-8
 track/sector, calculate
 absolute 5-13
 tracks, defective 5-13
 transfer address, disk 7-7
 transient portion 1-6
 types of devices
 block 2-5
 character 2-5

U

unique file, create 6-162
 init code 2-14
 inlock file access 6-165
 user stack 6-16
 using DOS functions 6-47

V

verify switch 6-92
 version specific information iii
 vertical position 3-9

W

WAIT, get a return code of a
 subprocess 6-151
 return code of a
 subprocess 6-151
 work areas 7-3
 wrap around 2-30
 write to a file or device 6-125
 write, random 6-79
 write, random block 6-85
 write, sequential 6-71

Numerals

0AH Buffered Keyboard
 Input 6-59
 0BH Check Standard Input
 Status 6-60
 0CH Clear Keyboard Buffer
 and Invoke a Keyboard
 Function 6-61
 0DH Disk Reset 6-62
 0EH Select Disk 6-63
 0FH Open File 6-64

- 00H Program Terminate 6-49
 01H Keyboard Input 6-50
 02H Display Output 6-51
 03H Auxiliary Input 6-52
 04H Auxiliary Output 6-53
 05H Printer Output 6-54
 06H Direct Console I/O 6-55
 07H Direct Console Input
 Without Echo 6-56
 08H Console Input Without
 Echo 6-57
 09H Print String 6-58
 1AH Set Disk Transfer
 Address 6-75
 1BH Allocation Table
 Information 6-76
 1CH Allocation Table
 Information for Specific
 Device 6-77
 10H Close File 6-65
 11H Search for First
 Entry 6-66
 12H Search for Next
 Entry 6-68
 13H Delete File 6-69
 14H Sequential Read 6-70
 15H Sequential Write 6-71
 16H Create File 6-72
 17H Rename File 6-73
 19H Current Disk 6-74
 2AH Get Date 6-88
 2BH Set Date 6-89
 2CH Get Time 6-90
 2DH Set Time 6-91
 2EH Set/Reset Verify
 Switch 6-92
 2FH Get Disk Transfer
 Address (DTA) 6-93
 2FH multiplex interrupt
 error codes 6-27
 function codes 6-27
 21H Random Read 6-78
 22H Random Write 6-79
 23H File Size 6-80
 24H Set Relative Record
 Field 6-81
 25H Set Interrupt Vector 6-82
 26H Create New Program
 Segment 6-83
 27H Random Block Read 6-84
 28H Random Block
 Write 6-85
 29H Parse Filename 6-86
 3AH Remove Subdirectory
 (RMDIR) 6-107
 3BH Change the Current
 Directory (CHDIR) 6-108
 3CH Create a File
 (CREAT) 6-109
 3DH (DOS 2.10) Open a
 File 6-110
 3DH (DOS 3.00 and 3.10)
 Open a File 6-112
 3EH Close a File
 Handle 6-122
 3FH Read from a File or
 Device 6-123
 30H Get DOS Version
 Number 6-94
 31H Terminate Process and
 Remain Resident 6-95
 33H Ctrl-Break Check 6-96
 35H Get Vector 6-97
 36H Get Disk Free Space 6-98
 38H (DOS 2.10) Return
 Country Dependent
 Information 6-99
 38H (DOS 3.00 and 3.10) Get
 or Set Country Dependent
 Information 6-101
 39H Create Subdirectory
 (MKDIR) 6-106
 4AH Modify Allocated
 Memory Blocks
 (SETBLOCK) 6-144

- 4BH Load or Execute a Program (EXEC) 6-145
- 4CH Terminate a Process (EXIT) 6-150
- 4DH Get Return Code of a Subprocess (WAIT) 6-151
- 4EH Find First Matching File (FIND FIRST) 6-152
- 4FH Find Next Matching File (FIND NEXT) 6-154
- 40H Write to a File or Device 6-125
- 41H Delete a File from a Specified Directory (UNLINK) 6-127
- 42H Move File Read Write Pointer (LSEEK) 6-128
- 43H Change File Mode (CHMOD) 6-130
- 44H I/O Control for Devices (IOCTL) 6-132
- 45H Duplicate a File Handle (DUP) 6-139
- 46H Force a Duplicate of a Handle (FORCDUP) 6-140
- 47H Get Current Directory 6-141
- 48H Allocate Memory 6-142
- 49H Free Allocated Memory 6-143
- 5AH (DOS 3.00 and 3.10) Create Unique File 6-162
- 5BH (DOS 3.00 and 3.10) Create New File 6-164
- 5CH (DOS 3.00 and 3.10) Lock/Unlock File Access 6-165
- 5E00H (DOS 3.10) Get Machine Name 6-168
- 5E02H (DOS 3.10) Set Printer Setup 6-169
- 5E03H (DOS 3.10) Get Printer Setup 6-170
- 5F02H (DOS 3.10) Get Redirection List Entry 6-171
- 5F03H (DOS 3.10) Redirect Device 6-173
- 5F04H (DOS 3.10) Cancel Redirection 6-176
- 54H Get Verify Setting 6-155
- 56H Rename a File 6-156
- 57H Get/Set a File's Date and Time 6-158
- 59H (DOS 3.00 and 3.10) Get Extended Error 6-160
- 62H (DOS 3.00 and 3.10) Get Program Segment Prefix Address (PSP) 6-178

