Appendix C

DOS Interrupts and Functions

Subdirectory access

RAM control

Device driver access

Time and date

DTA

Search directory

File access (FCB)

File access (handle)

Interrupt vectors

Disk/hard disk access

PSP access

DOS flag access

File information access

Country-specific functions

Other functions

Interrupt 2FH **Print spooler**

Interrupt 21H functions—arranged by function numbers

Interrupt 20H **DOS**
Terminate program **(Version 1 and up)**

Restores the three interrupt vectors whose contents were stored in the PSP before the program call, terminates the currently running program and returns control to MS-DOS. If the program redirected the vectors to its own routine, these vectors cannot be overwritten by another program. However, the terminating program releases the RAM it had occupied. Before turning control over to the calling program, this memory releases and all data buffers clear.

Input: CS = Segment address of the PSP

Output: No output

Remarks: COM programs automatically store the segment address of the PSP in the CS register. EXE programs require additional programming to load the segment address of the PSP into the CS register. Since the code and the PSP are stored in two separate segments, the address of the PSP must be loaded into the CS register. The code executes from another segment, which makes it impossible to call interrupt 32. To help overcome this problem, the value 0 and then the segment address of the PSP are pushed onto the stack. If a FAR RETURN command then executes, the program execution continues in the PSP segment at offset address 0. There a call for interrupt terminates the program.

For the first version of DOS, this interrupt is the usual method for ending a program. To terminate a program in DOS Version 2 and up, functions 31H or 4CH of DOS interrupt 21 H should be called instead.

Interrupt 21H, function 00H **DOS**
Terminate program **(Version 1 and up)**

Terminates execution of the currently running program and returns control to the calling program. Before this happens, the three interrupt vectors, whose contents had been stored in the PSP before the call of the program, are restored. If the program redirects these vectors to its own routine, they cannot be overwritten by another program. However, the terminating program does release the RAM it had occupied. Before turning control over to the calling program, the function releases this memory and clears all buffers.

Input: AH = 00H
CS = segment address of the PSP

Output: No output

Remarks: COM programs automatically store, in the CS register, the segment address of the PSP. Since the code and the PSP are stored in two separate segments, you cannot execute this function from an EXE program.

Instead of this function, use either function 31H or 4CH of interrupt 21H for terminating a program.

Interrupt 21H, function 01H **DOS**
Character input with echo **(Version 1 and up)**

Reads a character from the standard input device and displays it on the standard output device. When the function is called but a character doesn't exist, the function waits until a character is available. Since standard input and output can be redirected, this function is able to read a character from an input device other than the keyboard and send it to an output device other than the screen. The characters that are read may originate from other devices or from a file. If the character comes from a file, the input doesn't redirect to the keyboard once it reaches the end of the file. So, the function continues to try to read data from the file after it passes the end.

Input: AH = 01H

Output: AL = Character read

Remarks: If extended key codes are read, the function passes code 0 to the AL register. The function must be called again to read the actual code.

If the function encounters a <Ctrl><C> character (ASCII code 3), it calls interrupt 23H.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 02H **DOS**
Character output **(Version 1 and up)**

Displays a character on the standard output device. Since this device can be redirected, the character can be displayed on another output device or sent to a file. This function doesn't test whether or not the storage medium (disk or hard disk) is already full. Therefore, it will continue to try to write characters to this file.

Input: AH = 02H
DL = code of the character to be output

Output: No output

Remarks: Control codes such as backspace, carriage return and linefeed are executed when the function sends characters to the screen. If the output is redirected to a file, control codes are stored as normal ASCII codes.

If the function encounters a <Ctrl><C> character (ASCII code 3), it calls interrupt 23H.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 03H **DOS**
Read character auxiliary input **(Version 1 and up)**

Reads a character from the serial port. Access defaults to the device with the designation COM1, unless a MODE command previously redirected serial access.

Input: AH = 03H

Output: AL = Character received

Remarks: Since the serial port has no internal buffer, it can receive characters faster than it can read them. The unread characters are then ignored.

Before calling this function, communication parameters (baud rate, number of stop bits, etc.) must be set using the MODE command. Otherwise DOS defaults to 2400 baud, one stop bit, no parity and a word length of 8 bits.

The BIOS functions called from interrupt 14H are a more efficient way to access the serial port. Since they also allow reading of the serial port status, these functions offer more flexibility than the DOS functions.

If the function encounters a <Ctrl><C> character (ASCII code 3), it calls interrupt 23H.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 04H **DOS**
Auxiliary output **(Version 1 and up)**

Sends a character to the serial port. Unless a MODE command previously redirected serial access, access defaults to the device with the designation COM1.

Input: AH = 04H
DL = Character set for output

Output: No output

Remarks: As soon as the receiving device sends a signal to the function indicating that it is ready to receive it, the function transmits the character. Control then returns to the calling program.

Before calling this function, communication parameters (baud rate, number of stop bits, etc.) must be set using the MODE command.

Otherwise DOS defaults to 2400 baud, one stop bit, no parity and a word length of 8 bits.

The BIOS functions called from interrupt 14H are a more efficient way to access the serial port. Since they also allow reading of the serial port status, they offer more flexibility than the DOS functions.

If the function encounters a <Ctrl><C> character (ASCII code 3), it calls interrupt 23H.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 05H **DOS**
Character output to printer **(Version 1 and up)**

Sends a character to the printer. Access defaults to the device with the designation LPT1 (identical to PRN), unless a MODE command previously redirected printer access.

Input: AH = 05H
DL = Character code to be printed

Output: No output

Remarks: The function transmits the character only when the printer signals that it is ready to receive it. Then control returns to the calling program.

If the function encounters a <Ctrl><C> character (ASCII code 3), it calls interrupt 23H.

The BIOS functions called from interrupt 17H are more efficient for printer access. They offer more flexibility than the DOS printer functions for character output.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 06H **DOS**
Direct console I/O **(Version 1 and up)**

Reads characters from the standard input device and displays them on the standard output device. The read or written character isn't tested by the operating system (e.g., <Ctrl><C> has no effect on the program). Since standard input and output can be redirected, this function can read a character from an input device other than the keyboard and sends it to an output device other than the screen. The characters read may originate from other devices or from a file. When writing characters, this function doesn't test whether or not the storage medium (disk or hard disk) is

already full. Also, the calling program cannot determine whether all the characters have been read from an input file.

During character input, the function doesn't wait until a character is available. Instead, the function returns control to the calling program.

Input: AH = 06H
DL = 0–254: Send character code
DL = 255: Read a character

Output: Character output: No output
Character input: Zero flag=1: No character ready
Zero flag=0: Character read is in the AL register

Remarks: If extended key codes are read, the function passes code 0 to the AL register. The function must be called again to read the actual code.

ASCII code 255 (blank) cannot be displayed with this function because the function interprets ASCII code 255 as a command to input a character.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 07H **DOS**
Unfiltered character input without echo **(Version 1 and up)**

Reads a character from the standard input device without displaying the character on the standard output device. If a character doesn't exist when the function is called, the function waits until a character is available. The read character is not tested by the operating system (e.g., <Ctrl><C> has no effect on the program). Since standard input and output can be redirected, this function can read a character from an input device other than the keyboard. The characters that are read may originate from other devices or from a file. If the characters come from a file, the input doesn't redirect to the keyboard once it reaches the end of file. This causes the function to continue to try reading data from the file after it passes the end of file.

Input: AH = 07H

Output: AL = Character read

Remarks: If extended key codes are read, the function passes code 0 to the AL register. The function must be called again to read the actual code.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 08H **DOS**
Character input without echo **(Version 1 and up)**

Reads a character from the standard input device without displaying the character on the standard output device. If no character exists when the function is called, the function waits until a character is available.

Since standard input can be redirected, this function can read a character from an input device other than the keyboard. The characters read may originate from other devices or from a file. If the characters come from a file, the input doesn't redirect to the keyboard on reaching the end of file, so the function continues to try reading data from the file after it passes the end of file.

Input: AH = 08H

Output: AL = Character read

Remarks: If extended key codes are read, the function passes code 0 to the AL register. The function must be called again to read the actual code.

If the function encounters a <Ctrl><C> character (ASCII code 3), it calls interrupt 23H.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 09H **DOS**
Output character string **(Version 1 and up)**

Displays a character string on the standard output device. Since this device can be redirected, the character may be displayed on another output device or sent to a file. This function doesn't test whether or not the storage medium (disk or hard disk) is already full, and will continue to try to write the string to a file.

Input: AH = 09H
DS = String segment address
DX = String offset address

Output: No output

Remarks: The string must be stored in memory as a series of bytes which contain the ASCII codes of the characters to be output. A dollar sign character "$" (ASCII code 36) indicates, to DOS, the end of the string.

Control codes, such as backspace, carriage return and linefeed, are executed within the string.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 0AH **DOS**
Buffered input **(Version 1 and up)**

Reads a number of characters from the standard input device and transmits the characters to a buffer. The input ends when the user presses the <Return> key. The ASCII code of this key (13) is then placed in the buffer as the last character of the string.

Since standard input can be redirected, this function can read a character from an input device other than the keyboard. The characters read may originate either from other devices or from a file. If the characters come from a file, the input doesn't redirect to the keyboard on reaching the end of file, so the function continues to try reading data from the file after it passes the end.

Input: AH = 0AH
DS = Buffer segment address
DX = Buffer offset address

Output: No output

Remarks: The first byte of the buffer accepts the maximum number of characters (including the carriage return which ends the input) which can be read into the buffer, starting at memory location 2. In order to inform the function of the maximum number of characters it may read, this information must be entered, by the calling program, into the buffer before the function call.

After completion of the input, DOS places the number of characters read (excluding the carriage return) in memory location 1.

The buffer must be the number of the characters to be read plus 2 bytes.

When the input reaches the second to last memory location in the buffer, the computer beeps if you attempt to enter any character other than the <Return> key (end of input).

Extended key codes occupy two bytes in the buffer. The first byte contains the code 0, and the second byte contains the extended key code.

If the function encounters a <Ctrl><C> character (ASCII code 3), it calls interrupt 23H.

The <Backspace> and cursor keys let you edit the input without storing these keys in the buffer.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 0BH **DOS**
Get input status **(Version 1 and up)**

Determines whether a character is available for reading from the standard input device.

Input: AH = 0BH

Output: AL = 0: No character available
AL = 255: One or more characters available for reading

Remarks: If the function encounters a <Ctrl><C> character (ASCII code 3), it calls interrupt 23H.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 0CH **DOS**
Reset input buffer and then input **(Version 1 and up)**

Clears the input buffer then calls one of the character input functions. Since all the character input functions get their characters from the standard input device and standard input may redirected, this function only operates when the keyboard is the standard input device. In this case the characters could be entered before the function call but not read by a function. These existing characters are erased to ensure that the function call only reads characters which were inputted after its call.

Input: AH = 0CH
AL = Function to be called during call of function 10
DS = Input buffer segment address
DX = Input buffer offset address

Output: Functions 1, 6, 7 and 8: AL = Character to be read
Function 10: No output

Remarks: Functions 1, 6, 7, 8 and 10 can be passed to the function as calling functions.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 0DH **DOS**
Disk reset **(Version 1 and up)**

Sends all data stored in an internal DOS buffer to a block driver device (e.g., disk drive, hard disk). The open files (handles or FCBs) remain open.

Input: AH = 0DH

Output: No output

Remarks: Despite this function call, all open files must be closed in an orderly manner. Otherwise the current directory entry of the file may not update properly, which prevents access to new file data.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 0EH **DOS**
Select default disk drive **(Version 1 and up)**

Defines the the current default disk drive. Its designation appears as a prompt on the screen when the command interpreter expects input from the user. The drive indicated here will be used for all file access in which no special device was specified.

Input: AH = 0EH
DL = Drive number

Output: AL = Number of installed drives or volumes

Remarks: Drive A: has code number of 0, drive B: code number 1, etc.

Even if the PC has only one disk drive and one hard disk, the number of volumes in the AL register can be greater than two because the hard disk can be divided into multiple volumes. In addition, the PC can have one or more RAM disks as part of its configuration. For a PC with a single disk drive, you can only have two volumes because drive A: also simulates drive B:.

Unlike DOS Version 2, which permits 63 different device codes, DOS Version 3 permits 26 different devices (the letters A to Z). To keep compatibility between versions, limit your device access to a maximum of 26 devices.

BIOS interrupt 11H does a better job of reading the number of disk drives than this function.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 0FH **DOS**
Open file (FCB) **(Version 1 and up)**

Opens a file if one is available. After this function call executes successfully, the file can be read or written.

Input: AH = 0FH
DS = FCB segment address of the file
DX = FCB offset address of the file

Output: AL = 0: File found and opened
AL = 255: File not found

Remarks: Both normal and extended FCBs can be used.

If the file was found, DOS enters, into the FCB, the file size, the date and the time of its creation or last modification.

DOS sets the record length at 128 bytes. This record length can be changed in the FCB before opening a file. If you need a longer record length, the DTA must be moved (the original DTA is only 128 bytes long).

If random file access is performed, the random record field in the FCB must be set after the file opens successfully.

The file pointer points to the first byte of the file after the file opens.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 10H **DOS**
Close file (FCB) **(Version 1 and up)**

Writes all data currently in the DOS buffer to the file and closes the file. In addition, the directory entry changes to reflect the new file size and the date and time of the most recent modification to the file.

Input: AH = 10H
DS = FCB segment address of the file
DX = FCB offset address of the file

Output: AL = 0: File closed and directory entry revised
AL = 255: File not found in directory

Remarks: Only open files can be closed.

For disk files, the disk which was in the drive when the function call occurred must also be the disk that contains the file. Otherwise, the

function call writes an incorrect FAT and an incorrect directory to the disk, which makes the data that is already on the disk useless.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 11H **DOS**
Search for first match (FCB) **(Version 1 and up)**

Searches for the first occurrence in the disk directory of the filename indicated in the FCB.

Input: AH = 11H
DS = FCB segment address
DX = FCB offset address

Output: AL = 0: File found
AL = 255: File not found

Remarks: The FCB passed to the function contains the drive specifier and the filename for which the function should search.

The filename can contain the wildcard "?" to search for a group of files.

The search is made only in the current directory of the indicated device.

If the function searches for a normal file, a normal FCB can pass the information to the function. However, if you wish to search for a file with special attributes (volume name, subdirectories, hidden files, etc.), extended FCBs must be used.

If a file was found, the DTA contains an FCB of the same type as the FCBs. This FCB in the DTA contains the found filename. For this reason, the DTA must always be large enough to accept either a normal or an extended FCB.

The DTA can be switched to its own buffer using function 1AH, to ensure that it is large enough to accept the FCB.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 12H **DOS**
Search for next match (FCB) **(Version 1 and up)**

Searches for additional occurrences in the disk directory of the filename indicated in the FCB, after the file was found by function 17 (see above).

Input: AH = 12H
DS = FCB segment address
DX = FCB offset address

Output: AL = 0: File found
AL = 255: File not found (no other files available)

Remarks: This function can only be called after calling function 11H.

The FCB passed to the function contains the drive specifier and the filename for which the function should search.

If another filename was found its name is recorded in the FCB at the beginning of the DTA.

The DTA can be switched with function 1AH to its own buffer to ensure that it is large enough to accept the FCB.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 13H **DOS**
Delete file (FCB) **(Version 1 and up)**

Erases one or more files in the current directory of the specified device.

Input: AH = 13H
DS = FCB segment address
DX = FCB offset address

Output: AL = 0: file(s) erased
AL = 255: No file(s) found, or file(s) assigned Read Only attribute (undeletable)

Remarks: The FCB passed to the function contains both the device on which the files to be erased are located and the name of the file.

The filename can contain the wildcard "?" to erase a group of files.

Only files in the current directory of the indicated device may be erased.

If the function is used to delete a normal file, a normal FCB can pass the information to the function. However, if you want to delete a file with special attributes (volume name, subdirectories, hidden files, etc.), extended FCBs must be used.

Volumes may be deleted with this function; subdirectories may not.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 14H **DOS**
Sequential read (FCB) **(Version 1 and up)**

Reads the next sequential data block from a file.

Input: AH = 14H
DS = FCB segment address
DX = FCB offset address

Output: AL = 0: Block read
AL = 1: End of file reached
AL = 2: Segment wrap
AL = 3: Partial record read

Remarks: The function can only be called after the file was opened by the indicated FCB.

The DTA reads the block. If the DTA is not large enough, function 1AH must move the DTA into its own buffer.

The FCB records the size of the block and the corresponding number of bytes read.

Error 2 occurs when the DTA reaches the end of a segment and the block being read extends beyond the end of the segment.

Error 3 occurs when a partial block appears at the end of the file. The block is read in anyway and blank spaces bring the block up to the allocated block size.

After reading a block, the file pointer resets to the beginning of the next block so that the next function call automatically reads the next block.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 15H **DOS**
Sequential write (FCB) **(Version 1 and up)**

Writes a sequential block to a file.

Input: AH = 15H
DS = FCB segment address
DX = FCB offset address

Output: AL = 0: Block written
AL = 1: Medium (disk/hard disk) full
AL = 2: Segment overflow

Remarks: The function can only be called after the file was opened by the indicated FCB.

The DTA writes the block it contains to the file. If the DTA is not large enough to hold the file, function 1AH must be used to move the DTA into its own buffer.

The FCB records the size of the block and the corresponding number of bytes written.

Error 2 occurs if the DTA reaches the end of a segment and the block being written extends beyond the end of the segment.

After writing a block, the file pointer resets to the beginning of the next block, so that the next function call automatically writes the next block.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 16H **DOS**
Create or truncate file (FCB) **(Version 1 and up)**

Creates a new file, or dumps the contents of an existing file (file size=0 bytes). This function call allows other functions to read or write to the open file.

Input: AH = 16H
DS = FCB segment address
DX = FCB offset address

Output: AL = 0: File created or cleared
AL = 255: File could not be created (e.g., directory full)

Remarks: The contents of an existing file called by this function are lost.

After calling this function, the file is already open; you don't need to open the file using function 0FH (see above).

If you open the file using an extended FCB, you can assign certain attributes to the file (e.g., volume name, hidden file, etc.).

You cannot create a subdirectory using this function.

After opening the file, the file pointer moves to the first byte of the file.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 17H **DOS**
Rename file (FCB) **(Version 1 and up)**

Renames one or more files in the current directory of the specified device.

Input: AH = 17H
DS = FCB segment address
DX = FCB offset address

Output: AL = 0: File(s) renamed
AL = 255: No file found, or new filename matches old filename

Remarks: The FCB here is a special FCB, based on a normal FCB. The first 12 bytes contain the drive specifier and the name of the file to be renamed. However, this type of FCB has the new drive specifier and the new filename stored starting at memory location 10H. The drive specifier must be identical for both filenames.

The name of the file to be renamed can contain the wildcard "?", which renames several files. If the new filename contains the wildcard "?", the places in the filename and extension where a question mark appears in this parameter remain unchanged.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 19H **DOS**
Get default disk drive **(Version 1 and up)**

Returns the drive specifier of the default (current) disk drive.

Input: AH = 19H

Output: AL = Drive specifier

Remarks: This function identifies drive A as code 0, drive B as code 1, etc.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 1AH **DOS**
Set DTA address **(Version 1 and up)**

Transfers the DTA (Disk Transfer Area) to another area of memory. The DTA acts as buffer memory for all FCB supported file accesses.

Input: AH = 1AH
DS = New DTA segment address
DX = New DTA offset address

Output: No output

Remarks: This function must be called if the existing DTA has insufficient memory to handle the transmitted data.

When the program starts, MS-DOS places the DTA at address 128 in the PSP. Since the program starts after address 255 of the PSP, it is 128 bytes long.

DOS does not test the length of the DTA. Instead it assumes that the DTA is large enough to accept the transmitted data. If this is not the case, a DOS function can overwrite the excess data.

DOS recognizes an error during various functions if the DTA is at the end of a segment and the data to be transmitted exceeds the end of the segment.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 1BH **DOS**
Get allocation information for default drive **(Version 1 and up)**

Returns information about the format of the default drive.

Input: AH = 1BH

Output: AL = Number of sectors per cluster
DS = Media descriptor segment address
BX = Media descriptor offset address
DX = Number of clusters

Remarks: The media descriptor can return the following codes:

F8H:	Hard disk
F9H:	Disk drive: double-sided, 15 sectors per track (AT only)
FCH:	Disk drive: single-sided, 9 sectors per track
FDH:	Disk drive: double-sided, 9 sectors per track
FEH:	Disk drive: single-sided, 8 sectors per track
FFH:	Disk drive: double-sided, 8 sectors per track

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 1CH **DOS**
Get allocation information for specified drive **(Version 1 and up)**

Returns information about the format of the specified drive.

Input: AH = 1CH
DL = Drive specifier

Output: AL = Number of sectors per cluster
DS = Media descriptor segment address
BX = Media descriptor offset address
DX = Number of clusters

Remarks: This function identifies drive A as code 0, drive B as code 1, etc.

The media descriptor can return the following codes:

F8H:	Hard disk
F9H:	Disk drive: double-sided, 15 sectors per track (AT only)
FCH:	Disk drive: single-sided, 9 sectors per track
FDH:	Disk drive: double-sided, 9 sectors per track
FEH:	Disk drive: single-sided, 8 sectors per track
FFH:	Disk drive: double-sided, 8 sectors per track

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21(h), function 1DH **DOS**
Reserved **(Version 1 and up)**

Interrupt 21(h), function 1EH **DOS**
Reserved **(Version 1 and up)**

Interrupt 21(h), function 1FH **DOS**
Reserved **(Version 1 and up)**

Interrupt 21(h), function 20H **DOS**
Reserved **(Version 1 and up)**

Interrupt 21H, function 21H **DOS**
Random read (FCB) **(Version 1 and up)**

Reads a specified file record into the DTA.

Input: AH = 21H
DS = FCB segment address
DX = FCB offset address

Output: AL = 0: Record read
AL = 1: End of file reached
AL = 2: Segment overflow
AL = 3: Partial record read

Remarks: The function can only be called after the file was opened by the indicated FCB.

The record whose address is stored in the FCB starting at location 21H is read.

The DTA reads the record. If the DTA is not large enough, function 1AH must be called to move the DTA into its own buffer.

The FCB records the size of the record and the corresponding number of bytes read.

During the function call, the file pointer moves to the beginning of the record being read so that a subsequent call of a sequential read (function 14H—see above) reads the same record sequentially.

The record number does not increment following the function call, so a new call of this function would read the same record.

Error 2 occurs when the DTA reaches the end of a segment and the record being read extends beyond the end of the segment.

Error 3 occurs when a partial record appears at the end of the file. The record is read in anyway and blank spaces bring the record up to the allocated record size.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 22H **DOS**
Random write (FCB) **(Version 1 and up)**

Writes data from memory to the specified record in a file.

Input:
AH = 22H
DS = FCB segment address
DX = FCB offset address

Output:
AL = 0: record was written
AL = 1: Medium (disk/hard disk) full
AL = 2: segment overflow

Remarks:
The function can only be called after the file was opened by the indicated FCB.

The record whose address is stored in the FCB starting at location 21H is read.

The record is written from the DTA to the file. If the DTA is not large enough, function 1AH must move the DTA into its own buffer.

The FCB records the size of the record and the number of bytes read.

During the function call, the file pointer moves to the beginning of the record being read. This instructs subsequent calls of a sequential read (function 14H—see above) to read the same record sequentially.

The record number does not increment following the function call, so a new call of this function would read the same record.

Error 2 occurs when the DTA reaches the end of a segment and the record being written extends beyond the end of the segment.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 23H **DOS**
Get file size in records (FCB) **(Version 1 and up)**

Determines the size of a file based on the number of records in that file.

Input: AH = 23H
DS = FCB segment address
DX = FCB offset address

Output: AL = 0: Number of records found starting at FCB address 21H
AL = 255: File not found

Remarks: The FCB passed contains the drive specifier as well as the name and extension of the file to be examined.

Unlike the other FCB supported file accesses, the FCB requires the record size before the application can call this function.

A record size of 1 returns the size of the file in bytes.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 24H **DOS**
Set random record number **(Version 1 and up)**

Sets the record number in the FCB to the current position of the file pointer. Random access may begin at the point at which earlier sequential accesses left off.

Input: AH = 24H
DS = FCB segment address
DX = FCB offset address

Output: No output

Remarks: The function can only be called after the file was opened by the indicated FCB.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 25H **DOS**
Set interrupt vector **(Version 1 and up)**

Sets any interrupt vector to another routine.

Input: AH = 25H
AL = Interrupt number
DS = New interrupt routine segment address
DX = New interrupt routine offset address

Output: No output

Remarks: Before calling this function, the old contents of the interrupt vector to be changed should be read and stored using function 35H. After the program terminates, the old contents of the interrupt vector should be restored.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 26H **DOS**
Create PSP **(Version 1 and up)**

Copies the PSP (program segment prefix) of the executing program to a specified address in memory.

Input: AH = 26H
DX = New PSP segment address

Output: No output

Remarks: The new PSP offset address is 0.

DOS Version 1 uses this function to execute other programs by creating a PSP, loading the program after this PSP and executing it.

For DOS Version 2 up, use the EXEC function 4BH to load and execute additional programs instead of this function.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 27H **DOS**
Random block read (FCB) **(Version 1 and up)**

Reads one or more sequentially stored records into memory.

Input:
AH = 27H
CX = Number of records to be read
DS = FCB segment address
DX = FCB offset address

Output:
AL = 0: Record read
AL = 1: End of file reached
AL = 2: Segment overflow
AL = 3: Partial record read
CX = Number of records read

Remarks: The function can only be called after the file was opened by the indicated FCB.

The starting record is the record whose address is stored in the FCB, starting at location 21H.

The record data passes to the DTA. If the DTA is not large enough, function 1AH must move the DTA into its own buffer.

The FCB records the size of the record and the corresponding number of bytes read.

After the function call, the file pointer moves to the end of the last record that was read so that it points to the next record (following the last record read).

Error 2 occurs when the DTA reaches the end of a segment and the record being read extends beyond the end of the segment.

Error 3 occurs when a partial record appears at the end of the file. The record is read in anyway and blank spaces bring the record up to the allocated record size.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 28H **DOS**
Random block write (FCB) **(Version 1 and up)**

Writes one or more records in sequence to the specified file.

Input:
AH = 28H
CX = Number of records to be written
DS = FCB segment address
DX = FCB offset address

Output:
AL = 0: Record written
AL = 1: Medium (disk/hard disk) full
AL = 2: Segment overflow
CX = Number of records written

Remarks: The function can only be called after the file was opened by the indicated FCB.

The starting record is the record whose address is stored in the FCB starting at location 21H.

The FCB records the size of the record and the corresponding number of bytes read.

The data is written from the DTA to the file. If the DTA is not large enough, function 1AH must move the DTA into its own buffer.

After the function call, the file pointer moves to the end of the last record written so that it points to the next record, which follows the last record written. The record number increments by the number of records written.

Error 2 occurs when the DTA reaches the end of a segment and the record being written extends beyond the end of the segment.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 29H **DOS**
Parse filename to FCB **(Version 1 and up)**

Transfers an ASCII format filename into the proper fields of an FCB. The filename can include a drive specifier, filename and file extension.

Input:
AH = 29H
DS = Segment address of filename in memory
SI = Offset address of filename in memory
ES = FCB segment address
DI = FCB offset address

AL = Transmission parameters:
Bit 1 = 1: The drive specifier in the FCB changes only if the filename passed contains a drive specifier
0: The drive specifier changes anyway. If the filename passed contains no drive specifier, the the FCB defaults to 0 (current drive)
Bit 2 = 1: The filename in the FCB changes only if the filename parameter passed contains a filename
0: The filename changes. If the filename passed does not contain a filename, the filename in the FCB fills with spaces (ASCII code 32)
Bit 3 = 1: The file extension in FCB changes only if the filename passed contains an extension
0: The file extension in the FCB changes. If the filename passed has no extension, the extension field is padded with spaces (ASCII code 32)
Bits 4–8: Should contain the value 0

Output:
AL = 0: The filename passed contains no wildcards
AL = 1: The filename passed contains wildcards
AL = 255: Invalid drive specifier
DS = Segment address of the first character after parsed filename
SI = Offset address of the first character after parsed filename
ES = FCB segment address
DI = FCB offset address

Remarks:
The filename must end with an end character (ASCII code 0).

If the filename contains the wildcard "*", all corresponding fields in the FCB fill with the wildcard "?".

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 2AH **DOS**
Get system date **(Version 1 and up)**

Reads the current system date.

Input:
AH = 2AH

Output:
AL = Day of the week (0=Sunday, 1=Monday, etc.)
CX = Year
DH = Month
DL = Day

Remarks:
DOS calls the clock driver to read the date.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 2BH **DOS**
Set system date **(Version 1 and up)**

Sets the current system date as returned by function 2AH (see above).

Input: AH = 2BH
CX = Year
DH = Month
DL = Day

Output: AL = 0: O.K.
AL = 255: Date incorrect

Remarks: The date passes to the clock driver.

If the PC does not have a realtime clock, the date remains in effect until the PC is switched off or rebooted.

If the date entry is incorrect, the PC retains the old date.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 2CH **DOS**
Get system time **(Version 1 and up)**

Gets the current system time.

Input: AH = 2CH

Output: CH = Hours
CL = Minutes
DH = Seconds
DL = Hundredths of a second

Remarks: DOS calls the clock driver to read the time.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 2DH **DOS**
Set system time **(Version 1 and up)**

Sets the current system time.

Input: AH = 2DH
CH = Hours
CL = Minutes
DH = Seconds
DL = hundredths of a second

Output: AL = 0: O.K.
AL = 255: Incorrect time

Remarks: The time passes to the clock driver.

If the PC does not have a realtime clock, the time remains in effect until the PC is switched off or rebooted.

If the time entry is incorrect, the PC retains the old time.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 2EH **DOS**
Set verify flag **(Version 1 and up)**

Sets the verify flag. This flag determines whether data should be verified after a write operation to a block driver for proper transmission.

Input: AH = 2EH
DL = 0
AL = 0: Don't verify data
AL = 1: Verify data

Output: No output

Remarks: This flag can be controlled at the user level with the VERIFY ON and VERIFY OFF commands.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 2FH **DOS**
Get DTA address **(Version 2 and up)**

Returns the address of the DTA (Data Transmission Area), which serves as a data buffer for all FCB supported file accesses.

Input: AH = 2FH

Output: ES = DTA segment address
BX = DTA offset address

Remarks: This function determines the address of the DTA, but not the DTA's size.

After the start of a program, the DTA starts at memory location 128 of the PSP and has a length of 128 bytes.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 30H **DOS**
Get MS-DOS version number **(Version 2 and up)**

Returns the DOS version number.

Input: AH = 30H

Output: AL = Major version number (e.g., version 2.01=2)
AH = Minor version number (e.g., version 3.01=01)

Remarks: The major (whole) version number represents the number preceding the decimal point. For example, the version number 3.3 returns the major version number 3.

The minor (fractional) version number represents the number following the decimal point. It is always given as two digits. For example, Version 2.1 returns the minor version number 10 (0AH).

If the AL register contains a value of 0, the program runs under DOS Version 1. DOS Version 1.0 cannot use this function.

The contents of the DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 31H **DOS**
Terminate and stay resident **(Version 2 and up)**

Terminates the currently executing program and returns control to the calling program. The current program remains in memory for later recall.

Input: AH = 31H
AL = Return code
DX = Number of paragraphs to be reserved

Output: No output

Remarks: The return code in the AL register indicates whether or not the program called by it correctly executes. The calling program can read this number by calling function 77 (4DH). This value can be tested from within a batch file using the ERRORLEVEL and IF commands.

The number of 16-byte paragraphs to be reserved indicates how many bytes, beginning with the PSP, cannot be released for other uses.

Memory blocks reserved by function 48H are not affected by the value in the DX register because they can only be released by calling function 49H.

Interrupt 21H, function 33H, sub-function 0 **DOS**
Get <Ctrl><Break> flag **(Version 2 and up)**

Reads the <Ctrl><Break> flag. This determines whether DOS should test for active <Ctrl><C> or <Ctrl><Break> keys on each function call, or on character input/output calls. <Ctrl><C> and <Ctrl><Break> trigger interrupt 23H.

Input: AH = 33H
AL = 0

Output: DL = 0: Test only during character input/output
DL = 1: Test on every function call

Remarks: Since the <Ctrl><Break> flag is not part of the environment block of a program, it affects all programs which call the DOS character functions that test for <Ctrl><C> or the <Break> key.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 33H, sub-function 1 **DOS**
Set <Ctrl><Break> flag **(Version 2 and up)**

Sets and unsets the <Ctrl><Break> flag. This determines whether DOS should test for the activation of the <Ctrl><C> or <Ctrl><Break> keys on each DOS function call or character input/output calls. <Ctrl><C> and <Ctrl><Break> trigger interrupt 23H.

Input: AH = 33H
AL = 1
DL = 0: Test only during character input/output
DL = 1: Test on every function call

Output: No output

Remarks: Since the <Ctrl><Break> flag is not part of the environment block of a program, it affects all programs which call the DOS character functions that test for <Ctrl><C> or the <Break> key.

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 35H **DOS**
Get interrupt vector **(Version 2 and up)**

Returns the current contents of an interrupt vector and the address of the interrupt routine that belongs to it.

Input: AH = 35H
AL = Interrupt number

Output: ES = Interrupt routine segment address
BX = Interrupt routine offset address

Remarks: To ensure compatibility with future versions of DOS, instead of reading the vector's contents directly from the interrupt vector table, call this function for reading an interrupt vector.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 36H **DOS**
Get free disk space **(Version 2 and up)**

Returns information about the device (the block driver) from which the available memory space can be calculated.

Input: AH = 36H
DL = Device code

Output: AX = 65535: Device unavailable
AX< 65535: Number of sectors per cluster
BX = Number of available clusters
CX = Number of bytes per sector
DX = Total number of clusters on the device

Remarks: This function identifies drive A as code 0, drive B as code 1, etc.

The remaining memory on the medium can be computed from the number of bytes per sector multiplied by the number of sectors per cluster, multiplied by the number of free clusters.

The contents of the SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 38H **DOS**
Get country **(Version 2 and up)**

Determines country-specific parameters, which are set in the CONFIG.SYS file using the DOS COUNTRY command.

Input:
AH = 38H
AL = 0
DS = Buffer segment address
DX = Buffer offset address

Output: No output

Remarks: Before the function call, function 30H should be used to determine the DOS version. This can help the programmer compensate for differences between DOS versions during the call and return of this function.

The buffer must have at least 32 bytes allocated for recording the various country-specific parameters.

Following the function call, the individual bytes of this buffer contain the following information :

Bytes 0–1: Date format
0 = USA: Month-day-year
1 = Europe: day-month-year
2 = Japan: Year-month-day
Byte 2: ASCII code of the currency symbol
Byte 3: 0
Byte 4: ASCII code of the thousand character (comma/period)
Byte 5: 0
Byte 6: ASCII code of decimal character (period/comma)
Byte 7: 0
Bytes 8–31: reserved

The contents of the processor registers and the flag registers are not affected by this function.

Interrupt 21H, function 38H, sub-function 0 **DOS**
Get country **(Version 3 and up)**

Gets the country-specific parameters that are currently set.

Input:
AH = 38H
DS = Buffer segment address
DX = Buffer offset address
AL = 0: read current country parameters
AL = 1–254: Country code parameters to be read
AL = 255: Country code parameters to be read placed in the BX register

Output: Carry flag=0: O.K.
Carry flag=1: Invalid country code

Remarks: Before the function call, function 30H should be used to determine the DOS version. This can help the programmer compensate for differences between DOS versions during the call and return of this function.

The buffer must have at least 32 bytes allocated for recording the various country specific parameters.

Following the function call, the individual bytes of this buffer contain the following information:

Bytes 0–1: Date format
0 = USA: Month-day-year
1 = Europe: Day-month-year
2 = Japan: Year-month-day
Bytes 2–6: Currency indicator (string terminated by an end character)
Byte 7: ASCII code of the thousand character (comma/period)
Byte 8: 0
Byte 9: ASCII code of decimal character (period/comma)
Byte 10: 0
Byte 11: ASCII code of the date separation character
Byte 12: 0
Byte 13: ASCII code of the time separation character
Byte 14: 0
Byte 15: Currency format
bit 0 = 0: Currency symbol before the value
bit 0 = 1: Currency symbol after the value
bit 1 = 0: No spaces between value and currency symbol
bit 1 = 1: Space between value and currency symbol
Byte 16: Precision (number of decimal places)
Byte 17: Time format
bit 0 = 0: 12-hour clock
bit 0 = 1: 24-hour clock
Bytes 18-21: Address of character conversion routine (see below)
Bytes 22-33: reserved

Addresses 18 to 21 are the offset and segment addresses of a FAR procedure, which is used for accessing the country specific characters from the character set of the PC. The routine views the AL register's contents as the ASCII code of a lower case letter that should be converted to a capital letter. If a capital letter exists, it is retained in the AL register after the call. If the letter doesn't exist, the contents of the AL register remain unchanged. For example, the routine could be used to convert a lower case "a" into a capital "A".

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and the flag registers are not affected by this function.

Interrupt 21H, function 38H, sub-function 1 **DOS**
Set country **(Version 3 and up)**

Sets the current country-specific parameters. These parameters can be read using function 38H, sub-function 0. Previous versions of DOS required country-specific settings fromthe CONFIG.SYS file using the COUNTRY command. This function allows the user to set and change these parameters after booting.

Input:
AH = 38H
DX = 65535
AL = 1–254: Number of the country
AL > 254: Look in BX for country number
BX = Number of the country (if AL > 254)

Output:
Carry flag=0: O.K.
Carry flag=1: Invalid country code

Remarks:
Before the function call, function 30H should be used to determine that this command exists.

This function only allows setting of the country code, for which DOS has preset parameters. These parameters cannot be changed from this function.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 39H **DOS**
Create subdirectory **(Version 2 and up)**

Creates a new subdirectory on the specified device.

Input:
AH = 39H
DS = Subdirectory path segment address
DX = Subdirectory path offset address

Output:
Carry flag=0: Subdirectory created
Carry flag=1: Error (AX = error code)
AX=3: Path not found
AX=5: Access denied

Remarks:
The subdirectory path passed is an ASCII string which is terminated by an end character (ASCII code 0).

If the subdirectory path contains a drive specifier, the indicated device is accessed. Otherwise DOS creates the subdirectory on the current device.

An error can occur if any element of the path designation doesn't exist, a subdirectory already exists by that name, or the directory to be made is a subdirectory of the root directory and it is already filled.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 3AH **DOS**
Delete subdirectory **(Version 2 and up)**

Deletes a subdirectory from the specified drive.

Input: AH = 3AH
DS = Subdirectory path segment address
DX = Subdirectory path offset address

Output: Carry flag=0: Subdirectory deleted
Carry flag=1: Error (AX = error code)
AX=3: Path not found
AX=5: Access denied
AX=6: Directory to be deleted is the current directory

Remarks: The subdirectory path passed is an ASCII string which is terminated by an end character (ASCII code 0).

If the subdirectory path contains a drive specifier, the indicated device is accessed. Otherwise DOS deletes the subdirectory from the current device.

An error can occur if any element of the path designation doesn't exist, the subdirectory is the current directory, or the directory to be deleted still contains files.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 3BH **DOS**
Set current directory **(Version 2 and up)**

Sets the current subdirectory for the device indicated.

Input: AH = 3BH
DS = Subdirectory path segment address
DX = Subdirectory path offset address

Output: Carry flag=0: Subdirectory set
Carry flag=1: Error (AX = error code)
AX=3: Path not found

Remarks: The subdirectory path passed is an ASCII string which is terminated by an end character (ASCII code 0).

If the subdirectory path contains a drive specifier, the indicated device is accessed. Otherwise DOS deletes the subdirectory from the current device.

An error can occur if any element of the path designation doesn't exist.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 3CH **DOS**
Create or truncate file (handle) **(Version 2 and up)**

Creates a new file, or dumps the contents of an existing file (file size=0 bytes). This function call allows other functions to read or write to the open file.

Input:
AH = 3CH
CX = File attribute
Bit 0 = 1: File is read only
Bit 1 = 1: Hidden file
Bit 2 = 1: System file
DS = Filename segment address
DX = Filename offset address

Output:
Carry flag=0: O.K. (AX = file handle)
Carry flag=1: Error (AX = error code)
AX=3: Path not found
AX=4: No available handle
AX=5: Access denied

Remarks:
The various bits of the file attribute can be combined with each other.

The filename must be available as an ASCII string terminated by an end character (ASCII code 0). The filename parameter can contain a driver specifier, path, filename and extension. No wildcards are allowed. If you omit the drive specifier or path, DOS accesses the current drive or current directory.

An error can occur if any element of the path designation doesn't exist, if the file must be created in the root directory which is already full, or if a file with the same name already exists but cannot be cleared because it is write protected (bit 0 in the file attribute byte = 1).

If the function call executed successfully, all other handle functions can be called with this handle once the file opens.

The file pointer is set to the first byte of the file.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 3DH **DOS**
Open file (handle) **(Version 2 and up)**

Opens an existing file for access by other functions.

Input:
AH = 3DH
AL = Access mode
Bits 0–2: Read/write access
000(b) = File is read only
001(b) = File can only be written
010(b) = File can be read and written
Bit 3: 0(b)
Bits 4–6: File sharing mode
000(b) = Only current program can access the file (FCB mode)
001(b) = Only the current program can access the file
010(b) = Another program can read but not write the file
011(b) = Another program can write but not read the file
100(b) = Another program can read and write the file
Bit 7: Handle flag
0 = Child program of the current program can access file handle
1 = Current program can access file handle only
DS = Filename segment address
DX = Filename offset address

Output:
Carry flag=0: O.K. (AX = file handle)
Carry flag=1: Error (AX = error code)
AX=1:Missing file sharing software
AX=2: File not found
AX=3: Path not found or file doesn't exist
AX=4: No handle available
AX=5: Access denied
AX=12: Access mode not permitted

Remarks:
The filename must be available as an ASCII string terminated by an end character (ASCII code 0). The filename parameter can contain a driver specifier, path, filename and extension. No wildcards are allowed. If you omit the drive specifier or path, DOS accesses the current drive or current directory.

If the function call executes successfully, all other handle functions can be called with this handle once the file opens.

The file pointer is set to the first byte of the file.

DOS Version 2 uses only bits 0 to 2 of the access mode. All other bits, even under Version 3, should be 0 to ensure proper execution of the call.

DOS Version 3 uses the file sharing mode in bits 4 to 6 of the access mode only if the file is on a mass storage device which is part of a network. These three bits decide if and how the file, while it is open

using the current call, may be accessed by other programs from other PCs on the network.

Error 12 can occur only under DOS Version 3 and only within a network when the file is already opened by another program and if no other program can gain access to that file.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 3EH **DOS**
Close file (handle) **(Version 2 and up)**

Writes any data in the DOS buffers to a currently open file, then closes the file. If changes occur to the file, the new file size and the last date and time of modification are added to the directory.

Input: AH = 3EH
BX = Handle to be closed

Output: Carry flag=0: O.K.
Carry flag=1: Error (AX = error code)
AX=6: Unauthorized handle or file not opened

Remarks: Do not accidentally call this function with the numbers of the previous handle (the numbers 0 to 4) because the standard input device or standard output device may close. This would leave you unable to enter characters from the keyboard or display characters on the screen.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 3FH **DOS**
Read file or device (handle) **(Version 2 and up)**

Reads a certain number of characters by using a handle from a previously opened file or device and passes the characters to a buffer. The read operation starts at the current file pointer position.

Input: AH = 3FH
BX = File or device handle
CX = Number of bytes to be read
DS = Buffer segment address
DX = Buffer offset address

Output: Carry flag=0: O.K. (AX = number of bytes read)
Carry flag=1: Error (AX = error code)
AX=5: Access denied
AX=6: Illegal handle or file not open

Remarks: Characters can be read from a file or from a device (e.g., the standard input device [keyboard], which has the handle 0).

When the carry flag resets after the function call but the AX register has the value 0, this means that the file pointer has already reached the end of the file before the function call. So, no files could be read.

When the carry flag resets after the function call but the contents of the AX register are smaller than the contents of the CX register before the function call, this means that the desired number of bytes wasn't read because the end of the file was reached.

After the function call, the file pointer follows the last byte read.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 40H **DOS**
Write to file or device (handle) **(Version 2 and up)**

Writes a certain number of characters from a buffer to an open file or device by using a handle. The write operation begins at the file pointer's current position.

Input:
AH = 40H
BX = File or device handle
CX = Number of bytes to be written
DS = Buffer segment address
DX = Buffer offset address

Output:
Carry flag=0: O.K. (AX = number of bytes written)
Carry flag=1: Error (AX = error code)
AX=5: Access denied
AX=6: Illegal handle or file not open

Remarks: Characters can be written to a file or to a device (e.g., the standard output device [screen], which has the handle 1).

When the carry flag resets after the function call but the AX register has the value 0, this means that the file pointer has already reached the end of the file before the function call. Therefore no files could be written.

When the carry flag resets after the function call but the contents of the AX register are smaller than the contents of the CX register before the function call, this means that the desired number of bytes were not written because the end of file was reached.

After the function call, the file pointer follows the last byte written.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 41H **DOS**
Delete file (handle) **(Version 2 and up)**

Deletes the filename passed to the function. Through the call of this function, a file is erased and its name is passed to the function.

Input:
AH = 41H
DS = Filename segment address
DX = Filename offset address

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX = error code)
AX=2: File not found
AX=5: Access denied

Remarks: The filename must be available as an ASCII string terminated by an end character (ASCII code 0). The filename parameter can contain a drive specifier, path, filename and extension. No wildcards are allowed. If you omit the drive specifier or path, DOS accesses the current drive or current directory.

An error occurs when any element of the path designation doesn't exist or when the file has the attribute Read Only and therefore can not be written to or deleted. This attribute can be changed by using function 43H.

You cannot delete subdirectories or volume names with this function.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 42H **DOS**
Move file pointer (handle) **(Version 2 and up)**

Moves the file pointer of a previously opened file by using its handle. This allows random access because the individual records don't have to be read in sequence. The new file pointer position is given as an offset from the current position, either from the beginning of the file or from the end of the file. The offset itself is indicated as a 32-bit number.

Input:
AH = 42H
AL = Offset code
AL=0: Offset is relative to the beginning of the file
AL=1: Offset is relative to the current position of the file pointer
AL=2: Offset is relative to the end of the file
BX = Handle
CX = High word of the offset

DX = Low word of the offset

Output:
Carry flag=0: O.K.
DX = High word of the file pointer
AX = Low word of the file pointer
Carry flag=1: Error (AX = error code)
AX=1: Illegal offset code
AX=6: Illegal handle or File not open

Remarks: If offset codes 1 and 2 are accessed, negative offsets may be used to move the file pointer backwards or to place the pointer at the beginning of the file. It's possible to set the file pointer before the end of the file, which causes an error during the next read or write access to the file.

The position of the file pointer passed after the function call is always relative to the beginning of the file. The offset code used during the function call is independent of this file pointer position.

Passing offset code 2 and offset 0 returns the size of the file. This action moves the file pointer to the last byte of the file and the pointer's position returns to the calling program after the function call.

The contents of the BX, CX, , SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 43H, sub-function 0 **DOS**
Get file attributes **(Version 2 and up)**

Determines file attributes.

Input:
AH = 43H
AL = 0
DS = Filename segment address
DX = Filename offset address

Output:
Carry flag = 0: O.K. (CX = file attribute)
Bit 0=1: File can be read but not written
Bit 1=1: File hidden (not displayed on DIR)
Bit 2=1: File is a system file
Bit 3=1: File is the volume name
Bit 4=1: File is a subdirectory
Bit 5=1: File was changed since the last date/time
Carry flag = 1: Error (AX = error code)
AX=1: Unknown function code
AX=2: File not found
AX=3: Path not found

Remarks: The filename must be available as an ASCII string terminated by an end character (ASCII code 0). The filename parameter can contain a driver specifier, path, filename and extension. No wildcards are allowed. If you

omit the drive specifier or path, DOS accesses the current drive or current directory.

An error occurs when any element of the path designation or the file does not exist.

The contents of the BX, CX, , SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 43H, sub-function 1 **DOS**
Set file attributes **(Version 2 and up)**

Sets the file attributes.

Input:
AH = 43H
AL = 1
CX = File attributes
Bit 0 = 1: File can be read but not written
Bit 1 = 1: File hidden (not displayed on DIR)
Bit 2 = 1: File is a system file
Bit 3 = 0
Bit 4 = 0
Bit 5 = 1: File was changed since the last date/time
DS = Filename segment address
DX = Filename offset address

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX = error code)
AX=1: Unknown function code
AX=2: File not found
AX=3: Path not found
AX=5: Attribute cannot be changed

Remarks:
The filename must be available as an ASCII string terminated by an end character (ASCII code 0). The filename parameter can contain a driver specifier, path, filename and extension. No wildcards are allowed. If you omit the drive specifier or path, DOS accesses the current drive or current directory.

An error occurs when any element of the path designation or the file does not exist.

Neither subdirectories nor volume names can be accessed with this function. For this reason bits 3 and 4 of the file attribute must be 0 during the function call. If you attempt to access a subdirectory or a volume name anyway, the function returns error code 5.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 0 **DOS**
IOCTL: Get device information **(Version 2 and up)**

Permits access of a character driver's device attribute.

Input:
AH = 44H
AL = 0
BX = Handle

Output:
Carry flag=0: O.K. (DX = device attribute)
Bit 14= 1: Processes control characters through IOCTL
Bit 7 = 1: Character driver
Bit 5 = 0: Cooked mode operation
1: Raw mode operation
Bit 3 = 1: Clock driver operation
Bit 2 = 1: NUL driver operation
Bit 1 = 1: Console output driver (screen)
Bit 0 = 1: Console input driver (keyboard)
Carry flag=1: Error (AX = error code)
AX=1: Unknown function code
AX=6: Handle not opened or does not exist

Remarks:
A handle is passed (not the name of the addressed character driver which must be connected with this driver). This can be one of the five pre-assigned handles (0 to 4). A handle could have been previously opened for a certain device with the help of the Open function (function 3DH), and then passed to the function. For example, since the standard input and output devices (handles 0 and 1) can be redirected, this method assures that the indicated device is accessed.

If bit 7 in the device attribute is unequal to 1, the driver addressed is not a character driver and the significance of the individual bits in the device attribute disagrees with those of the device driver.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 1 **DOS**
IOCTL: Set device information **(Version 2 and up)**

Sets the character device attributes.

Input:
AH = 44H
AL = 1
BX = Handle
CX = Number of bytes written
DX = Device attributes
Bit 14= 1: Processes control characters through IOCTL using sub-functions 2 and 3
Bit 7 = 1: Character driver

Bit 5 = 0: Cooked mode operation
Bit 5 = 1: Raw mode operation
Bit 3 = 1: Clock driver operation
Bit 2 = 1: NUL driver operation
Bit 1 = 1: Console output driver (screen)
Bit 0 = 1: Console input driver (keyboard)

Output: Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=1: Unknown function code
AX=6: handle not opened or handle does not exist

Remarks: A handle is passed but it is not the name of the addressed character device, which must be connected with this device. This can be one of the five pre-assigned handles (0 to 4). A handle could have previously been opened, with the Open function, for a certain device and then passed to the function. For example, since the standard input and output devices (handles 0 and 1) can be redirected, this method assures that the indicated device is accessed.

To change various device attribute bits with this function, use sub-function 0 to read the device attributes first. Then this sub-function can reset the device attribute bits in the device driver.

If bit 7 in the device attribute is unequal to 1, the driver addressed is not a character driver. The meanings of the individual bits in the device attribute disagree with those in the device driver.

This function is especially useful for switching between cooked mode and raw mode within a character driver (bit 5).

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 2 **DOS**
IOCTL: Read data from character device **(Version 2 and up)**

Reads data from a character device. This function defines the number of bytes of data to read from the buffer, which contains the data taken from the character device.

Input: AH = 44H
AL = 2
BX = Handle
CX = Number of bytes to be read
DS = Buffer segment address
DX = Buffer offset address

Output: Carry flag=0: O.K. (AX = Number of bytes sent)
Carry flag=1: Error (AX = Error code)
AX=1: Unknown function code
AX=6: Handle not opened or does not exist

Remarks: A handle is passed, but it is not the name of the addressed character device which must be connected with this device. This can be one of the five pre-assigned handles (0 to 4). A handle could have previously been opened with the Open function (function number 3DH) for a certain device, then passed to the function. For example, since the standard input and output devices (handles 0 and 1) can be redirected, this method assures that the indicated device is accessed.

An error always occurs if the handle passed is connected with a block driver instead of a character driver.

The driver defines the data type and structure.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 3 **DOS**
IOCTL: Send data to character device **(Version 2 and up)**

Sends data from an application program directly to a character device. The calling function defines the number of bytes to be transferred from a buffer to the device.

Input: AH = 44H
AL = 3
BX = Handle
CX = Number of bytes to be transmitted
DS = Buffer segment address
DX = Buffer offset address

Output: Carry flag=0: O.K.
AX = Number of bytes sent
Carry flag=1: Error (AX = Error code)
AX=1: Unknown function code
AX=6: Handle not opened or does not exist

Remarks: A handle is passed, but it is not the name of the addressed character device which must be connected with this device. This can be one of the five pre-assigned handles (0 to 4). A handle could have previously been opened with the Open function (function number 61) for a certain device, then passed to the function. For example, since the standard input and output devices (handles 0 and 1) can be redirected, this method assures that the indicated device is accessed.

An error always occurs if the handle passed is connected with a block driver instead of a character driver.

The driver defines the data type and structure.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 4 **DOS**
IOCTL: Read data from block device **(Version 2 and up)**

Reads data for an application directly from a block device. The calling function defines the number of bytes to be copied by the device into a buffer.

Input:
AH = 44H
AL = 4
BX = Device designation
CX = Number of bytes to be read
DS = Buffer segment address
DX = Buffer offset address

Output:
Carry flag=0: O.K.
AX = Number of bytes sent
Carry flag=1: Error (AX = Error code)
AX=1: Unknown function code
AX=15: Unknown device

Remarks: Instead of defining the device driver, the device designation parameter defines the device from which data will be received. Code 0 represents device A:, 1 represents device B:, etc.

The driver defines the data type and structure.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 5 **DOS**
IOCTL: Send data to block device **(Version 2 and up)**

Sends data from an application program directly to a character device. The calling function defines the number of bytes to be transferred from a buffer to the device.

Input:
AH = 44H
AL = 5
BX = Device designation
CX = Number of bytes to be sent
DS = Buffer segment address
DX = Buffer offset address

Output:
Carry flag=0: O.K.
AX = Number of bytes sent
Carry flag=1: Error (AX = Error code)
AX=1: Unknown function code

AX=15: Unknown device

Remarks: Instead of defining the device driver, the device designation parameter defines the device from which data will be received. Code 0 represents device A:, 1 represents device B:, etc.

The driver defines the data type and structure.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 6 **DOS**
IOCTL: Read input status **(Version 2 and up)**

Determines whether a device driver can transmit data to an application program.

Input: AH = 44H
AL = 6
BX = Handle

Output: Carry flag=0: O.K. (AX = Input status)
AX=0: Driver not ready
AX=255: Driver ready
Carry flag=1: Error (AX = Error code)
AX=1: Unknown function code
AX=5: Access denied

Remarks: The handle passed can refer to either a character driver or a file.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 7 **DOS**
IOCTL: Read output status **(Version 2 and up)**

Determines whether a device driver can receive data from an application program.

Input: AH = 44H
AL = 7
BX = Handle

Output: Carry flag=0: O.K. (AX = Output status)
AX=0: Driver is not ready
AX=255: Driver is ready
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function number
AX=5: Access denied

Remarks: The handle passed can refer to either a character driver or a file.

If the handle refers to a file, the block device driver signals its readiness to receive data, even if the medium containing the file is full and no additional data can be appended to the end of the file.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 8 **DOS**
IOCTL: Test for changeable block device **(Version 3 and up)**

Determines whether the block device medium (e.g., disk, hard disk, etc.) can be changed.

Input:
AH = 44H
AL = 8
BL = Device designation

Output:
Carry flag=0: O.K. (AX=status code)
AX = 0: Medium changeable
AX = 1: Medium unchangeable
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function number
AX=15: Invalid drive number

Remarks: The device designation parameter defines the device being addressed instead of the device driver. Code 0 represents device A:, 1 represents device B:, etc.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 9 **DOS**
IOCTL: Test for local or remote drive **(Version 3.1 and up)**

Determines whether a drive (block device) is local (part of the PC making the inquiry) or remote (part of another PC in a network).

Input:
AH = 44H
AL = 9
BL = Device designation

Output:
Carry flag=0: O.K.
DX = device attribute
Bit 12 = 0: Local
Bit 12 = 1: Remote
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function number
AX=15: Invalid drive specification

Remarks: You can access this sub-function only if networking software has previously been installed.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 0AH **DOS**
IOCTL: Test for local or remote handle **(Version 3.1 and up)**

Determines whether a file associated with this handle is local (part of the PC making the inquiry) or remote (part of another PC in a network).

Input:
AH = 44H
AL = 0AH
BX = Handle

Output:
DX = IOCTL code
Bit 15 = 0: Local
Bit 15 = 1: Remote
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function number
AX=6: Handle not opened or does not exist

Remarks: You can access this sub-function only if networking software has previously been installed.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 44H, sub-function 0BH **DOS**
IOCTL: Change retry count **(Version 3 and up)**

Sets the variables that specify the number of attempts at file access. One PC within a network may try to access a file that is already being accessed by another PC. The PC attempting access repeats the file access procedure the number of times and the number of waiting periods defined by these variables.

Input:
AH = 44H
AL = 0BH
BX = Number of attempts
CX = Waiting time between attempts

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function number

Remarks: You can only access this sub-function if networking software has previously been installed.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 45H **DOS**
Duplicate handle **(Version 2 and up)**

Creates a duplicate of the handle passed. This duplicate handle interfaces with the same file or device as the first handle. If the first handle refers to a file, the value of the first handler's file pointer joins with the file pointer of the duplicate handle.

Input: AH = 45H
BX = Handle

Output: Carry flag=0: O.K. (AX = the new handle
Carry flag=1: Error (AX = Error code)
AX=4: No additional handle available
AX=6: Handle not opened or does not exist

Remarks: Without having to close the file, this function updates a file directory entry after its modification. A file can be closed using function 62 (3EH).

If the file pointer of one of the two handles changes position due to the call of a read or write function, the other file pointer also changes automatically.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 46H **DOS**
Force duplicate of handle **(Version 2 and up)**

Refers a second file handle to the save device or file as the first file handle. The second handle's file pointer also contains the same value as the first handle's file pointer.

Input: AH = 46H
BX = First handle
CX = Second handle

Output: Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=4: No additional handle available
AX=6: Handle not opened or does not exist

Remarks: If the function call connects the second handle to an open file, the file closes before the forced duplication.

If the file pointer of one of the handles changes position due to the call of a read or write function, the other file pointer also changes automatically.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 47H **DOS**
Get current directory **(Version 2 and up)**

Gets an ASCII string listing the complete path designation of the current directory of the indicated device. This string passes to the specified buffer.

Input:
AH = 47H
DL = Device designation
DS = Buffer segment address
SI = Buffer offset address

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX=Error code)
AX=15: Invalid drive specification

Remarks:
The device designation parameter defines the device being addressed instead of the device driver. Code 0 represents the current device, 1 represents device A:, etc.

The path description in the buffer terminates with an end character (ASCII code 0). This description has no drive specifier or \ character (root directory specifier). If the root directory is the current directory, the end character becomes the first character in the buffer.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 48H **DOS**
Allocate memory **(Version 2 and up)**

Reserves an area of memory for program use.

Input:
AH = 48H
BX = Number of paragraphs to be reserved

Output:
Carry flag=0: O.K.
AX=Memory area segment address)
Carry flag=1: Error (AX = Error code)
AX=7: Memory control block destroyed
AX=8: Insufficient memory
BX = Number of paragraphs available

Remarks:
A paragraph consists of 16 bytes.

If memory allocation was successfully executed, the allocated range begins at address AX:0000.

This function always fails when executed from within a COM program because the PC assigns the total amount of free memory to a COM program when it executes.

The contents of the CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 49H **DOS**
Release memory **(Version 2 and up)**

Releases memory previously allocated by function 72 (49H—see above) for any purpose.

Input: AH = 49H
ES = Memory area segment address

Output: Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=7: Memory control block destroyed
AX=9: Incorrect memory area passed in ES

Remarks: Since DOS knows the size of the memory area to be released, no parameter exists for passing memory size.

If the wrong segment address appears in the ES register during the function call, memory assigned to another program can be released. This can lead to a system crash or other consequences.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 4AH **DOS**
Modify memory allocation **(Version 2 and up)**

Changes the size of a memory area previously reserved using function 72 (3FH—see above).

Input: AH = 4AH
BX = New memory area size in paragraphs
ES = Memory area segment address

Output: Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=7: Memory control block destroyed
AX=8: Insufficient memory
BX = Number of paragraphs available

Remarks: A paragraph has 16 bytes.

If the wrong segment address appears in the ES register during the function call, memory assigned to another program can be released. This can lead to a system crash or other consequences.

Since the PC assigns the total amount of free memory to a COM program when it executes, this function call always fails when executed from within a COM program.

COM programs should use this function to release all unnecessary memory since all RAM becomes part of a COM program. This is especially important before calling the EXEC function (function number 75 (4BH).

The contents of the CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 4BH, sub-function 0 **DOS**
Execute program **(Version 2 and up)**

Executes another program from within a program and continues execution of the original program after the called program finishes its run. The function requires the name of the program to be executed and the address of a parameter block, which contains information that is important to the function.

Input:
AH = 4BH
AL = 0
ES = Parameter block segment address
BX = Parameter block offset address
DS = Program name segment address
DX = Program name offset address

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function number
AX=2: Path or program not found
AX=5: Access denied
AX=8: Insufficient memory
AX=10: Wrong environment block
AX=11: Incorrect format

Remarks: The directory name passed is an ASCII string which is terminated by an end character (ASCII code 0). It can contain a path designation and drive specifier. No wildcards are allowed. If no drive specifier or path designation exists, the function accesses the current drive or directory.

Only EXE or COM programs can be executed. To execute a batch file, the command processor (COMMAND.COM) must be called using the /c parameter followed by the name of the batch file.

The parameter block must have the following format:

Bytes 0–1:	Environment block segment address
Bytes 2–3:	Command parameter offset address
Bytes 4–5:	Command parameter segment address
Bytes 6–7:	First FCB offset address
Bytes 8–9:	First FCB segment address
Bytes 10–11:	Second FCB offset address
Bytes 12–13:	Second FCB segment address

If the segment address of the environment block is a 0, the called program has the same environment block as the calling program.

The command parameters must be stored so that the parameter string begins with a byte representing the number of characters in the command line. Next follow the individual ASCII characters, which are terminated by a carriage return (ASCII code 13) (this carriage return is not counted as a character).

The first FCB passed is copied to the PSP of the called program starting at address 5CH. The second FCB passed is copied to the PSP of the called program starting at address 6CH. If the called program does not obtain information from the two FCBs, any desired value can be entered into the FCB fields at the parameter block.

After the call of this function, all registers are destroyed except the CS and IP registers. For later recall, save their contents before the function call.

The program called should have all the handles available to the calling program.

Interrupt 21H, function 4BH, sub-function 3 **DOS**
Execute overlay **(Version 2 and up)**

Loads a second program into memory as an overlay without automatically executing the second program.

Input:
AH = 4BH
AL = 3
ES = Parameter block segment address
BX = Parameter block offset address
DS = Program name segment address
DX = Program name offset address

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function number
AX=2: Path or program not found
AX=5: Access denied
AX=8: Insufficient memory

AX=10: Wrong environment block
AX=11: Incorrect format

Remarks: The directory name passed is an ASCII string which is terminated by an end character (ASCII code 0). It can contain a path designation and drive specifier. No wildcards are allowed. If no drive specifier or path designation exists, the function accesses the current drive or directory.

Only EXE or COM programs can be executed. To execute a batch file, the command processor (COMMAND.COM) must be called using the /c parameter followed by the name of the batch file.

The parameter block must have the following format:

Byte 0–1:	Segment address where the overlay will be stored (offset address=0)
Byte 2–3:	Relocation factor

The relocation factor requires the value 0 for COM programs. Use the segment address at which the program should load when accessing EXE programs.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 4CH **DOS**
Terminate with return code **(Version 2 and up)**

Terminates a program and passes an end code for which function 77 (4DH-see below) searches. This function releases the memory previously occupied by the terminated program.

Input: AH = 4CH
AL = Return code

Output: No output

Remarks: This function may be used for program termination instead of the other functions listed earlier.

This function call restores the contents of the three interrupt vectors that were stored in the PSP when the program started execution.

Before passing control to the calling program, all handles opened by this program close, along with the corresponding files. This is not applicable to files accessed using FCBs.

A batch file can test for the return code using the ERRORLEVEL and IF batch commands.

Interrupt 21H, function 4DH **DOS**
Get return code **(Version 2 and up)**

Checks a program, called from another program by the EXEC function, for the return code passed by the called program when it terminates.

Input: AH = 4DH

Output: AH = Type of program termination
AH=0: Normal end
AH=1: End through <Ctrl><C> or <Break>
AH=2: Device access error
AH=3: Call of function 49 (31H)
AL = Return code

Remarks: This function reads the return code of the called program only once.

The contents of the AX, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and flag registers are not affected by this function. The contents of all other registers may change.

Interrupt 21H, function 4EH **DOS**
Search for first match **(Version 2 and up)**

Searches for the first occurrence of the filename listed. The file can have certain attributes, so a search can be made through subdirectories and volume names.

Input: AH = 4EH
CX = File attribute
DS = Filename segment address
DX = Filename offset address

Output: Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=2: Path not found
AX=18: No file with the attribute found

Remarks: The directory name passed is an ASCII string which is terminated by an end character (ASCII code 0). It can contain a path designation and drive specifier. No wildcards are allowed. If no drive specifier or path designation exists, the function accesses the current drive or directory.

The search defaults to normal files (attribute 0). Any set attribute bits extends the search to normal files and any other file types.

If a matching file occurs, the first 43 bytes of the DTA contain the following information about this file:

Bytes 0–20: Reserved
Byte 21: File attribute
Bytes 22–23: Time of last modification to file

Bytes 24–25: Date of last modification to file
Bytes 26–27: Low word of file size
Bytes 28–29: High word of file size
Bytes 30–42: ASCII filename and extension terminated by an end character (ASCII code 0)

This function may only be called to search for the first occurrence of a file. If you want to search for a group of files using wildcards, function 4FH (see below) must be called.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 4FH **DOS**
Search for next match (handle) **(Version 2 and up)**

Searches for subsequent occurrences of the filename listed after function 78 (above) executed successfully.

Input: AH = 4FH

Output: Carry flag=0: O.K.
Carry flag=1: Error (AX=Error code)
AX=18: No other files found with this attribute

Remarks: If a matching file occurs, the first 43 bytes of the DTA contain the following information about this file:

Bytes 0–20: Reserved
Byte 21: File attribute
Bytes 22–23: Time of last modification to file
Bytes 24–25: Date of last modification to file
Bytes 26–27: Low word of file size
Bytes 28–29: High word of file size
Bytes 30–42: ASCII filename and extension terminated by an end character (ASCII code 0)

This function can only be called if function 4EH has been called once and if the DTA remains unchanged.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 54H **DOS**
Get verify flag **(Version 2 and up)**

Gets the current status of the verify flag. This flag determines whether or not data transmitted to a medium (floppy disk or hard disk) should be verified after the transmission.

Input: AH = 54H

Output: AL = Verify flag
AL=0: Verify off
AL=1: Verify on

Remarks: Function 2EH (see above) controls the status of the verify flag.

The contents of the AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES and flag registers are not affected by this function.

Interrupt 21H, function 56H **DOS**
Rename file (handle) **(Version 2 and up)**

Renames a file or moves the file to another directory of a block device. Moving is possible only within the different directories of one particular device (i.e., you can't move a file from a hard disk directory to a floppy disk directory).

Input: AH = 56H
DS = Old filename segment address
DX = Old filename offset address
ES = New filename segment address
DI = New filename offset address

Output: Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=2: File not found
AX=3: Path not found
AX=5: Access denied
AX=11: Not the same device

Remarks: The directory name passed is an ASCII string which is terminated by an end character (ASCII code 0). It can contain a path designation and drive specifier. No wildcards are allowed. If no drive specifier or path designation exists, the function accesses the current drive or directory.

An error occurs if you attempt to move the file to a filled root directory.

This function cannot access subdirectories or volume names.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 57H, sub-function 0 **DOS**
Get file date and time **(Version 2 and up)**

Gets the date and time of the creation or last modification of a file.

Input:
AH = 57H
AL = 0
BX = Handle

Output:
Carry flag=0: O.K.
CX=Time
DX=Date
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function
AX=6: Invalid handle

Remarks: In order for it to be accessed with a handle, the file must have been previously opened or created using one of the handle functions.

The time appears in the CX register in the following format:

Bits 0–4:	Seconds in 2-second increments
Bits 5–10:	Minutes
Bits 11–15:	Hours

The date appears in the DX register in the following format:

Bits 0–4:	Day of the month
Bits 5–8:	Month
Bit 9–15:	Year (relative to 1980)

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 57H, sub-function 1 **DOS**
Set file date and time **(Version 2 and up)**

Stores the date and time of the creation or last modification of a file in the corresponding file and device.

Input:
AH = 57H
AL = 1
BX = Handle
CX = Time
DX = Date

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function
AX=6: Invalid handle

Remarks: In order to be accessed with a handle, the file must have been previously opened or created using one of the handle functions.

The time appears in the CX register in the following format:

Bits 0–4: Seconds in 2-second increments
Bits 5–10: Minutes
Bits 11–15: Hours

The date appears in the DX register in the following format:

Bits 0–4: Day of the month
Bits 5–8: Month
Bit 9–15: Year (relative to 1980)

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 58H, sub-function 0 **DOS**
Get allocation strategy **(Version 3 and up)**

Determines the method currently in use by MS-DOS for allocating blocks of memory. If a program allocates memory using function 48H, different programs in memory may already have memory blocks assigned to them. Since these requested memory blocks vary in size, DOS has three methods of allocating memory to a program:

- First fit: DOS starts searching at the start of memory and allocates the first memory block it finds of the requested size;
- Best fit: DOS searches all available memory blocks and allocates the smallest suitable memory block it finds (the most efficient method);
- Last fit: DOS starts searching at the end of memory and allocates the first memory block it finds of the requested size.

Input: AH = 58H
AL = 0

Output: Carry flag=0: O.K.
AX=0: First fit (start from beginning of memory)
AX=1: Best fit (search for best-fitting memory block)
AX=2: Last fit (start from end of memory)
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function number

Remarks: The allocation strategy applies to all programs.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 58H, sub-function 1 **DOS**
Set allocation strategy **(Version 3 and up)**

Defines the method currently in use by MS-DOS for allocating blocks of memory. If a program allocates memory using function 48H, different programs in memory may already have memory blocks assigned to them. Since these requested memory blocks vary in size, DOS has three methods of allocating memory to a program:

- First fit: DOS starts searching at the start of memory and allocates the first memory block it finds of the requested size;
- Best fit: DOS searches all available memory blocks and allocates the smallest suitable memory block it finds (the most efficient method);
- Last fit: DOS starts searching at the end of memory and allocates the first memory block it finds of the requested size.

Input:
AH = 58H
AL = 1
BX = Allocation strategy
BX=0: First fit (start from beginning of memory)
BX=1: Best fit (search for best-fitting memory block)
BX=2: Last fit (start from end of memory)

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function number

Remarks:
The allocation strategy applies to all programs.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 59H **DOS**
Get extended error information **(Version 3 and up)**

Gets information about errors that occur during the call of one of the functions of either interrupt 21H or interrupt 24H. This information includes detailed information about the error, its origin and the action the user should take to alleviate the error.

Input: AH = 59H
BX = 0

Output: AX = Description of error
BH = Cause of error
BL = Recommended action
CH = Source of error

Remarks: The following codes describe the error:

Code	Error
0:	No error
1:	Invalid function number
2:	File not found
3:	Path not found
4:	Too many files open at once
5:	Access denied
6:	Invalid handle
7:	Memory control block destroyed
8:	Insufficient memory
9:	Invalid memory address
10:	Invalid environment
11:	Invalid format
12:	Invalid access code
13:	Invalid data
14:	Reserved
15:	Invalid drive
16:	Current directory cannot be removed
17:	Different device
18:	No additional files
19:	Medium write protected
20:	Unknown device
21:	Device not ready
22:	Unknown command
23:	CRC error
24:	Bad request structure length
25:	Seek error

Code	Error
26:	Unknown medium type
27:	Sector not found
28:	Printer out of paper
29:	Write error
30:	Read error
31:	General failure
32:	Sharing violation
33:	Lock violation
34:	Unauthorized disk change
35:	FCB not available
80:	File already exists
81:	Reserved
82:	Directory cannot be created
83:	Terminate after call of interrupt 24H

The following codes describe the cause of the error:

Code	Error
1:	No memory available on the medium
2:	Temporary access problem–may end soon
3:	Access unauthorized
4:	Internal error in system software
5:	Hardware error
6:	Software failure not caused by running application program
7:	Application program error
8:	File not found
9:	Invalid file format/type
10:	File locked
11:	Wrong medium in drive, bad disk or medium problem
12:	Other error

The following codes describe the action needed to fix the error:

Code	Error
1:	Repeat process several times, then ask user to abort/ignore
2:	Repeat process several times pausing each time, then ask user to abort/ignore
3:	Ask user for correct information (e.g., filename)
4:	Terminate program as completely as possible
5:	Terminate program NOW (no file closing, etc.)
6:	Ignore error
7:	Ask user to remove error source and repeat process

The following codes describe the source of the error:

Code	Error
1:	Unknown
2:	Block device (disk drive, hard disk, etc.)
3:	Network
4:	Serial device
5:	RAM

The contents of the CS, DS, SS and ES registers are not affected by this function. All other register contents are destroyed.

Interrupt 21H, function 5AH **DOS**
Create temporary file (handle) **(Version 3 and up)**

Creates a temporary file in memory for storage during program execution. The filename doesn't matter because the access occurs through the assigned handle. Since this function allows several files open at the same time, DOS creates filenames from the current date and time. Every temporary file is ensured its own particular name because the function cannot be called more than once at a time.

Input:
AH = 5AH
CX = File attribute
DS = Directory segment address
DX = Directory offset address

Output:
Carry flag=0: O.K.
AX=Handle
DS=Complete filename segment address
DX=Complete filename offset address
Carry flag=1: Error (AX = Error code)
AX=3: Path not found
AX=5: Access denied

Remarks:
The directory name passed is an ASCII string which is terminated by an end character (ASCII code 0). It can contain a path designation and drive specifier. No wildcards are allowed. If no drive specifier or path designation exists, the function accesses the current drive or directory.

The bits of the file attribute have the following meanings:

Bit 0 = 1: Read only file
Bit 1 = 1: Hidden file
Bit 2 = 1: System file

Temporary files are not automatically deleted after program execution. The file must be closed using function 3EH, then the temporary file must be deleted using function 41H.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function.

Interrupt 21H, function 5BH **DOS**
Create new file (handle) **(Version 3 and up)**

Creates a file in the specified directory based upon an ASCII file format. If no drive specifier or path is provided, the file opens in the default (current) directory.

Input:
AH = 5BH
CX = File attributes:
CX=00: Normal file
CX=01: Read-only file
CX=02: Hidden file
CX=04: System file
DS = ASCII file specification segment address
DX = ASCII file specification offset address

Output:
Carry flag=0 (AX= file handle)
Carry flag=1 (AX = Error code)
AX=3: Path not found
AX=4: No handle available
AX=5: Access denied
AX=80 (50H): File already exists

Remarks: An error occurs when any element of the path designation doesn't exist, when the filename already exists in the specified directory, or when an attempt is made to create the file in an already full root directory.

The file defaults to the normal read/write attribute, which allows both read and write operations. This attribute can be changed by using function 43H.

Interrupt 21H, function 5CH **DOS**
Control record access **(Version 3 and up)**

Locks or unlocks a particular section of a file. This function operates on multitasking and networking systems.

Input:
AH = 5CH
AL = Function code
AL=00: Lock file section
AL=01: Unlock file section
BX = File handle
CX = High word of section offset
DX = Low word of section offset
SI = High word of section length
DI = Low word of section length

Output: Carry flag=0: Successful lock/unlock
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function code
AX=6: Invalid handle
AX=33 (21H): All or part of section already locked

Remarks: This function can only be used on files already opened or created using functions 3CH, 3DH, 5AH or 5BH.

The corresponding call to unlock a file region must contain the identical file offset and file region length.

Interrupt 21H, function 5EH, sub-function 0 **DOS**
Get machine name **(Version 3.1 and up)**

Returns the address of an ASCII string which defines the local computer type within a network.

Input: AH = 5EH
AL = 00
DS = User buffer segment address
DX = User buffer offset address

Output: Carry flag=0: Successful execution
CH = 00: Name undefined
CH > 00: Name defined
CL = NETBIOS name number (when CH<>00)
DS = Identifier segment address (when CH<>00)
DX = Identifier offset address (when CH<>00)
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function code

Remarks: The computer type is a 15-byte-long string terminated by an end character (ASCII code 0).

Interrupt 21H, function 5EH, sub-function 2 **DOS**
Set printer setup **(Version 3.1 and up)**

Specifies a string which precedes all output to a particular printer used by a network. This string allows network users to assign their own individual printing parameters to the shared printer.

Input: AH = 5EH
AL = 02
BX = Redirection list index (see Remarks below)
CX = Printer setup string length
DS = Printer setup string segment address
SI = Printer setup string offset address

Output: Carry flag=0: Successful execution
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function code

Remarks: The contents of register BX (redirection list index) come from function 94 5EH, sub-function 2. Function 5EH, sub-function 3 (see below) can supply the current printer setup string.

Interrupt 21H, function 5EH, sub-function 3 **DOS**
Get printer setup **(Version 3.1 and up)**

Gets the printer setup string assigned to a particular network printer by using function 5EH, sub-function 2 (see above).

Input: AH = 5EH
AL = 03
BX = Redirection list index)
DS = Setup string receiving buffer segment address
SI = Setup string receiving buffer offset address

Output: Carry flag=0: Successful execution
CX=Printer setup string length
ES=Segment address of buffer retaining setup string
DI=Offset address of buffer retaining setup string
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function code

Remarks: The contents of register BX (redirection list index) come from function 5EH, sub-function 2. Function 5EH, sub-function 3 can supply the current printer setup string.

Interrupt 21H, function 5FH, sub-function 2 **DOS**
Get redirection list entry **(Version 3.1 and up)**

Gets the system redirection list. This list assigns local names to network printers, files or directories.

Input: AH = 5FH
AL = 02
BX = Redirection list index (see Remarks below)
DS = Device name buffer segment address (16 bytes)
SI = Device name buffer offset address (16 bytes)
ES = Network name buffer segment address (128 bytes)
DI = Network name buffer offset address (128 bytes)

Output: Carry flag=0: Successful execution
BH = Status flag
0: Valid device
1: Invalid device
BL = Device type
3: Printer
4: Drive
BP = Destroyed
CX = Parameter value in memory
DX = Destroyed
DS = ASCII format local device name segment address
SI = ASCII format local device name offset address
ES = ASCII format network name segment address
DI = ASCII format network name offset address
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function code
AX=18: No more files available

Remarks: The contents of register CX come from function 5FH, sub-function 3 (see below).

Interrupt 21H, function 5FH, sub-function 3 **DOS**
Redirect device **(Version 3 and up)**

Redirects device access in a network, assigning a network name to a local device.

Input: AH = 5FH
AL = 03
BL = Device type
BL=3: Printer
BL=4: Drive
CX = Parameter value in memory
DS = ASCII format local device name segment address
SI = ASCII format local device name offset address
ES = ASCII format network name and password segment address
DI = ASCII format network name and password offset address

Output: Carry flag=0: Successful execution
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function code; string format incorrect; device redirected
AX=3: Path not found
AX=5: Access denied
AX=8: Insufficient memory

Remarks: The contents of register CX are supplied from function 5FH, sub-function 3.

Device names can be drive specifiers (e.g., A:), printer names (i.e., LPT1, PRN, LPT2 or LPT3) or null strings. If you enter a null string and pass-

word as the device name, DOS tries to open access to the network using the password.

Interrupt 21H, function 5FH, sub-function 4 **DOS**
Cancel redirection **(Version 3 and up)**

Disables the current redirection by removing local name assignments to network printers, files or directories.

Input: AH = 5FH
AL = 04
BX = Redirection list index (see Remarks below)
DS = ASCII format local device name segment address
SI = ASCII format local device name offset address

Output: Carry flag=0: Successful execution
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function code; device name not on network
AX=15: Redirection halted

Remarks: Device names can be drive specifiers (e.g., A:), printer names (i.e., LPT1, PRN, LPT2 or LPT3) or strings beginning with double backslashes (i.e., \\). A string preceded by two backslashes terminates communications between the local computer and the network.

Interrupt 21H, function 62H **DOS**
Get PSP address **(Version 3 and up)**

Gets the segment address of the PSP from the currently executing program.

Input: AH = 62H

Output: BX = PSP segment address

Remarks: The PSP starts at address BX:0000.

The contents of the AX, CX, DX, SI, DI, BP, CS, DS, SS, ES registers and the flag registers are not affected by this function.

Interrupt 21H, function 63H, sub-function 0 **DOS**
Get lead byte table **(Version 2.25 only)**

Gets the address of the system table which defines the byte ranges for the PC's extended character sets.

Input: AH = 9963H
AL = 00: Get address of system lead byte table

Output: DS = Table segment address
SI = Table offset address

Remarks: This function is available only in DOS Version 2.25.

Interrupt 21H, function 63H, sub-function 1 **DOS**
Set or clear interim console flag **(Version 2.25 only)**

Clears the interim console flag.

Input: AH = 63H
AL = 01: Clear or set interim console flag
DL = Interim console flag setting
DL=01: Set interim console flag
DL=00: Clear interim console flag

Output: No output

Remarks: This function is available only in DOS Version 2.25.

Interrupt 21H, function 63H, sub-function 2 **DOS**
Get interim console flag **(Version 2.25 only)**

Gets the interim console flag.

Input: AH = 63H
AL = 02: Get interim console flag value

Output: DL = Flag value

Remarks: This function is available only in DOS Version 2.25.

Interrupt 21H, function 64H **DOS**
Reserved **(Version 3 and up)**

Interrupt 21H, function 65H **DOS**
Get extended country information **(Version 3.3 and up)**

Gets information about the specific country/code page.

Input: AH = 65H
AL = sub-function:
AL = 1: Get international information

AL = 2: Get uppercase pointer table
AL = 4: Get pointer to uppercase pointer table (filename)
AL = 6: Get pointer to collation table
BX = Code page:
BX = -1: active CON device
CX = Length of buffer allocated to receive information
DX = Country ID number
DX = -1: Default
ES:DI = Address of buffer allocated to receive information

Output:
Carry flag=0: Successful execution
Carry flag=1: Error (AX = Error code)

Remarks:
The information this function returns is an extended version of the information returned by int 21H, function 38H.

An error may occur if the country code in DX is invalid, or if the code page number is different from the country code, or if the buffer length specified in the CX register is less than five bytes. If the buffer is not long enough to receive all the information, the function accepts as much information as the buffer will accept. This buffer contains the following information after the call:

Byte 0: ID code for information
Bytes 1-2: Length of buffer
Bytes 3-4: Country ID
Bytes 5-6: Code page
Bytes 7-8: Date format
0 = USA: Month-day-year
1 = Europe: Day-month-year
2 = Japan: Year-month-day
Bytes 9-13: Currency indicator
Bytes 14-15: ASCII code of the thousand character (comma/period)
Bytes 16-17: ASCII code of the decimal character (period/comma)
Bytes 18-19: ASCII code of the date separation character
Bytes 20-21: ASCII code of the time separation character
Byte 22: Currency format
bit 0 = 0: Currency symbol before the value
bit 0 = 1: Currency symbol after the value
bit 1 = 0: No spaces between value and currency symbol
bit 1 = 1: Space between value and currency symbol
Byte 23: Precision (number of decimal places)
Byte 24: Time format
bit 0 = 0: 12-hour clock
bit 0 = 1: 24-hour clock
Bytes 25-28: Address of character conversion routine
Bytes 29-30: ASCII data separator
Bytes 31-40: Reserved

Interrupt 21H, function 66H **DOS**
Get or set code page **(Version 3.3. and up)**

Gets or sets the current code page.

Input: AH = 66H
AL = sub-function:
AL = 1: Get code page
AL = 2: Select code page
BX = Selected code page (if AL = 2)

Output: Carry flag=0: Successful execution
If AL =1 used for input:
BX = active code page
DX = default code page
Carry flag=1: Error (AX = Error code)

Remarks: If sub-function 2 is used, COUNTRY.SYS supplies the code page number.

The DEVICE... (CONFIG.SYS), NLSFUNC and MODE CP PREPARE commands (AUTOEXEC.BAT) must have already configured the system for code page switching before this function may be called.

Interrupt 21H, function 67H **DOS**
Set handle count **(Version 3.3 and up)**

Sets the maximum number of accessible files and devices that may be currently opened using handles.

Input: AH = 67H
BX = Number of handles desired

Output: Carry flag=0: Successful execution
Carry flag=1: Error (AX = Error code)

Remarks: The PSP's default table reserved for the process can control 20 handles.

An error occurs if the content of the BX register is greater than 20, or if insufficient memory exists to allocate a block for the extended table.

If the number in the BX register is greater than the number of entries assigned by the FILES entry in the CONFIG.SYS file, no error occurs. However, attempts at opening a file or device fail if all file entries are in use, even if file handles are still available.

Interrupt 21H, function 68H **DOS**
Commit file **(Version 3.3 and up)**

Writes all DOS buffers associated to a specific handle to the specified device. If the handle points to a file, the file's contents, date and size are updated.

Input: AH = 68H
BX = File handle

Output: Carry flag=0: Successful execution
Carry flag=1: Error (AX = Error code)

Remarks: This function performs the same task as closing and reopening a file or duplicate handle, even without handles. If this function accesses a character device's handle, the carry flag returns 0 but nothing else happens.

Multiprocessing and networking applciations maintain control of the file.

Interrupt 22H **DOS**
Terminate address **(Version 1 and up)**

Contains the address of a routine which terminates a program. Control returns to the program that called for termination. You should never call this routine directly.

DOS stores the contents of this interrupt vector in the PSP of the program to be executed before passing control to the program. This prevents program changes to the vector, which could prevent DOS from calling the termination routine.

Interrupt 23H **DOS**
<Ctrl><C> handler address **(Version 1 and up)**

Contains the address of a routine which executes when the user presses <Ctrl><C> or <Ctrl><Break>. You should never directly call this routine.

DOS stores the contents of this interrupt vector in the PSP of the program to be executed before passing control to the program. This prevents program changes to the vector, which could prevent DOS from calling the termination routine.

Interrupt 24H **DOS**
Critical error handler address **(Version 1 and up)**

Represents a routine called during hardware access (e.g., disk drive) when a critical error occurs. You should never directly call this routine.

When an application routine is called during a critical error, bit 7 of the AH register indicates the type of failure (0 = disk/hard disk error, 1= other errors). A disk/hard disk error will only be reported after several attempted accesses. During the call, the DI register receives one of the following codes:

0: Disk write protected
1: Access on unknown device
2: Drive not ready
3: Invalid command
4: CRC error
5: Bad request structure length
6: Seek error
7: Unknown device type

8: Sector not found
9: Printer out of paper
10: Write error
11: Read error
12: General failure

The error routine restores the SS, SP, DS, ES, BX, CX and DX registers to the same values that they contained during the call. During execution it can only access functions 1 to 0CH of interrupt 21H. It should be terminated by an IRET instruction and pass one of the following codes to the AL register:

0: Ignore error
1: Repeat the operation
2: Terminate program using interrupt 23H
3: Fail system call (Version 3 and up only)

If a program changes the content of this interrupt vector, the program can terminate without restoring the memory contents. Since RAM can be released and used by other programs, the critical error routine can be overwritten by another program in memory. When this occurs, a critical error could cause a system crash because a completely different code now exists at the location of the old error handler routine.

Before passing control to the program, DOS stores the contents of this interrupt vector in the PSP of the program to be executed. This prevents program changes to the vector, which could prevent DOS from calling the termination routine. During program termination, the contents of the interrupt vector pass from the PSP to the vector; then the system calls the routine.

Interrupt 25H **DOS**
Absolute disk read **(Version 1 and up)**

Reads one or more consecutive sectors from a disk or hard disk.

Input:
AL = Drive specifier
CX= Number of sectors to read
DX= First sector to read
DS= Buffer segment address
BX= Buffer offset address

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=1: Bad command
AX=2: Bad address
AX=4: Sector not found
AX=8: DMA error
AX=16: CRC error
AX=32: Disk controller error
AX=64: Seek error
AX=128: Device does not respond

Remarks: In the AL register 0 represents drive A:, 1 represents drive B:, etc.

All the sectors of the medium can be accessed. DOS itself uses this interrupt to read the root directory and the FAT of a medium. The data are read from the medium into the buffer of the calling program. After the function call, the contents of all registers, except the segment register, may change.

After the interrupt call, the stack pointer changes position because two bytes stored on the stack during the call are removed and not returned. These bytes represent the flag register, which can be read from the stack using the POPF instruction. The old value of the stack pointer can be set by adding 2 to its contents. If you omit the stack pointer correction, the stack could overflow. Because of this, you cannot call this interrupt from higher level languages. You must call it from assembly language.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function. The contents of all other registers may change.

Interrupt 26H **DOS**
Absolute disk write **(Version 1 and up)**

Writes one or more consecutive sectors to a disk or hard disk.

Input:
AL = Device designation
CX = Number of sectors to be written
DX = First sector to be written
DS = Buffer segment address
BX = Buffer offset address

Output:
Carry flag=0: O.K.
Carry flag=1: Error (AX = Error code)
AX=1: Bad command
AX=2: Bad address
AX=3: Medium write protected
AX=4: Sector not found
AX=8: DMA error
AX=16: CRC error
AX=32: Disk controller error
AX=64: Seek error
AX=128: Device does not respond

Remarks: In the drive specifier 0 represents drive A:, 1 represents drive B:, etc.

All the sectors of the medium can be accessed. DOS itself uses this interrupt to write the root directory and the FAT to a medium. The data are written from the buffer of the calling program to the medium. After the function call, the contents of all registers, except the segment register, may change.

After the interrupt call, the stack pointer changes position because two bytes stored on the stack during the call are removed and not returned. These bytes represent the flag register, which can be read from the stack using the POPF instruction. The old value of the stack pointer can be set by adding 2 to its contents. If you omit the stack pointer correction, the stack could overflow. Because of this, you cannot call this interrupt from higher level languages. You must call it from assembly language.

The contents of the BX, CX, DX, SI, DI, BP, CS, DS, SS and ES registers are not affected by this function. The contents of all other registers may change.

Interrupt 27H **DOS**
Terminate and stay resident **(Version 1 and up)**

Terminates the currently executing program and returns control to the program that called the current program. Unlike other functions used for program termination, the memory used by the current program keeps the program code for later recall.

Input: CS = PSP segment address
DX = Number of bytes + 1 to be reserved

Output: No output

Remarks: This function is only suitable for calling COM programs.

The number of bytes to be reserved relates to the beginning of the PSP.

The value in the DX register has no effect on memory blocks reserved by function 48H of interrupt 21H.

An error occurs during the call of this interrupt if the value in the DX register ranges from FFF1H to FFFFH.

This interrupt does not close open files.

Interrupt 2FH, sub-function 0 **DOS**
Get print spool install status **(Version 3 and up)**

Gets current installation status of the print spooler.

Input: AH = 2FH
AL = 0

Output: Carry flag=0: Successful execution
AL = 0: O.K. to install
AL = 1: Don't install
AL = 255: Already installed
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function
AX=2: File not found
AX=3: Path not found

AX=4: Too many files currently open
AX=5: Access denied
AX=8: Print queue full
AX=9: Print spooler busy
AX=12: Name too long
AX=15: Invalid drive

Interrupt 2FH, sub-function 1 **DOS**
Send file to print spooler **(Version 3 and up)**

Passes a file to the print spooler.

Input:
AH = 2FH
AL = 1
DS = Print packet (see below) segment address
DX = Print packet (see below) offset address

Output:
Carry flag=0: Successful execution
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function
AX=2: File not found
AX=3: Path not found
AX=4: Too many files currently open
AX=5: Access denied
AX=8: Print queue full
AX=9: Print spooler busy
AX=12: Name too long
AX=15: Invalid drive

Remarks: The five-byte print packet contains print spooler information. The first byte indicates the DOS version (0=Versions 3.1 to 3.3); the remaining bytes indicate the segment and offset addresses of the file specification.

Interrupt 2FH, sub-function 2 **DOS**
Remove file from print queue **(Version 3 and up)**

Deletes a file from the print spooler queue.

Input:
AH = 2FH
AL = 2
DS = ASCII-format file segment address
DX = ASCII-format file offset address

Output:
Carry flag=0: Successful execution
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function
AX=2: File not found
AX=3: Path not found
AX=4: Too many files currently open
AX=5: Access denied
AX=8: Print queue full

AX=9: Print spooler busy
AX=12: Name too long
AX=15: Invalid drive

Remarks: This sub-function allows wildcards (? and *) in file specifications, allowing you to delete more than one file at a time from the print queue.

Interrupt 2FH, sub-function 3 **DOS**
Cancel all files in print queue **(Version 3 and up)**

Cancels all files waiting in the print spooler queue for printing.

Input: AH = 2FH
AL = 3

Output: Carry flag=0: Successful execution
Carry flag=1: Error (AX = Error code)
AX=1: Invalid function
AX=2: File not found
AX=3: Path not found
AX=4: Too many files currently open
AX=5: Access denied
AX=8: Print queue full
AX=9: Print spooler busy
AX=12: Name too long
AX=15: Invalid drive

Interrupt 2FH, sub-function 4 **DOS**
Hold print jobs for status check **(Version 3 and up)**

Halts all print jobs while testing for spooler status.

Input: AH = 2FH
AL = 4

Output: Carry flag=0: Successful execution
Carry flag=1: Error
DX = Number of errors
DS = Print queue segment address
SI = Print queue offset address

Remarks: The print queue segment and offset addresses point to a set of 64-byte filenames in the queue. Each entry contains an ASCII file specification.

The first filename in the queue is the file currently printing in the print spooler. The last filename in the queue has a zero in the first byte of the specification.